Actively Learning Deep Neural Networks with Uncertainty Sampling Based on Sum-Product Networks

Mohamadsadegh Khosravani
Department of Computer Science
University of Regina
mko041@uregina.ca

Sandra Zilles
Department of Computer Science
University of Regina
sandra.zilles@uregina.ca

Abstract

Active learning is a popular approach for reducing the amount of data in training deep neural network models. Its success hinges on the choice of an effective acquisition function, which ranks not yet labeled data points according to their expected informativeness. In uncertainty sampling, the uncertainty that the current model has about a point's class label is the main criterion for this type of ranking. This paper proposes a new approach to uncertainty sampling in training a Convolutional Neural Network (CNN). The main idea is to use the feature representation extracted by the CNN as data for training a Sum-Product Network (SPN). Since SPNs are typically used for estimating the distribution of a dataset, they are well suited to the task of estimating class probabilities that can be used directly by standard acquisition functions such as max entropy and variational ratio. Moreover, we enhance these acquisition functions by weights calculated with the help of the SPN model; these weights make the acquisition function more sensitive to the diversity of conceivable class labels for a data point. The effectiveness of our method is demonstrated in an experimental study on the MNIST, Fashion-MNIST and CIFAR-10 datasets, where we compare it to the state-of-the-art methods MC Dropout and Bayesian Batch.

1 Introduction

Deep neural networks have become the dominant approach for learning from high-dimensional data. A major problem, however, is the large amount of data required for training them. Obtaining highly accurate predictors with fewer examples is the main motivation behind active learning [Settles, 2009], in which the learning method actively selects unlabeled data points whose labels are then acquired. The idea is to select data points that are expected to be particularly informative, and thus to reduce the number of data points required for producing an accurate predictor. Uncertainty-based approaches consider a point informative if its class label is highly uncertain given the current training data [Lewis and Gale, 1994]. By contrast, distribution-based methods intuitively assume that points are informative if they lie in a relatively dense area (according to some underlying data distribution) from which not many training points have been sampled yet [Yi et al., 2022].

When applying uncertainty-based methods in deep learning, one challenge is how to estimate uncertainty. Gal and Ghahramani [2016] proposed a method for estimating the uncertainty of the label of a point x, for a convolutional neural network (CNN) trained on a given dataset. Their method, called MC Dropout, uses random dropout to create many (similar but not identical) variants of the CNN and then averages the probabilities output by these variants on point x, in order to get a robust estimate of the posterior probability of any given class label for x. This probability estimate can then be fed into a standard uncertainty function like entropy or variational ratio in order to assess

Preprint. Under review.

uncertainty. MC Dropout showed very good performance on some image datasets, but it was recently reported that it is not very effective on more complex datasets like CIFAR-10 [Pinsler et al., 2019].

The main contribution of this paper is to propose SPN-CNN, a new method for estimating posterior probabilities of class labels, using an approach that is more effective than MC Dropout on complex datasets. The main idea behind our method is to deploy the probabilistic nature of Sum-Product Networks (SPNs, Poon and Domingos [2011]). SPNs are deep graphical models that handle probabilities by design, and their typical purpose is to produce estimates of the data distribution. Their structure carries more intuitive semantics than that of neural networks. Because of their probabilistic nature, we consider SPNs particularly suited to the task of deriving probabilities needed for uncertainty-based deep active learning. In a nutshell, SPN-CNN trains a CNN and then use its extracted data representation in order to train an SPN which is used to assess the probabilities with which the training data allows us to assign unseen data points to classes. As in MC Dropout, SPN-CNN feeds these probability estimates into standard functions to calculate uncertainty. Since uncertainty-based active learning is prone to selecting highly correlated data points [Sener and Savarese, 2017], we further use the SPN to calculate weights by which to discount uncertainty for the sake of diversity.

In an experimental analysis, we compare SPN-CNN to two state-of-the-art methods, namely MC Dropout [Gal and Ghahramani, 2016] and Bayesian Batch [Pinsler et al., 2019], using two standard uncertainty-based acquisition functions (Max Entropy and Variational Ratio) as well as three standard benchmark datasets for image processing (MNIST, CIFAR-10, Fashion-MNIST). It turns out that SPN-CNN outperforms MC Dropout in all cases except when deployed with Max Entropy on the least complex dataset (MNIST). It also outperforms Bayesian Batch in all settings, except when a particularly large initial training data set is provided on the CIFAR-10 data. (Arguably though, active learning is more critical when initial training datasets are not very large.) In the sum, our approach to train an SPN on the data representation extracted by a CNN is highly effective in estimating probabilities for active learning.

2 Related Work

In classical machine learning, active learning for classification is well-studied [Settles, 2009]. One main aspect in the design of an active learning system is the criterion by which the active learner selects data points for labeling. To this end, the two main approaches are uncertainty sampling (choosing a point for which the class label is most uncertain in the current model) [Lewis and Gale, 1994, Nguyen et al., 2022] and diversity sampling (choosing points to reflect the underlying distribution of the data) [Brinker, 2003]. In uncertainty sampling, standard measures of uncertainty are, for example, Shannon Entropy [Shannon, 1948] and Variational Ratio [Freeman, 1965]; both of these measures need estimates of the probability that a data point x has class label c, given the training data. Such "class probability" estimates can be derived in various ways, e.g., from the output of a neural network after applying softmax, or from the proximity of a point to the decision boundary when using a linear separator like an SVM [Brinker, 2003].

Recently, deep active learning has become a very active branch of research [Ren et al., 2021, Zhan et al., 2022], driven mostly by the need to reduce the typically large amounts of data required to train deep classification models. Deep model structures to which active learning has been applied include stacked Restricted Boltzmann Machines [Wang and Shang, 2014], variational adversarial networks [Sinha et al., 2019], as well as CNNs [Wang et al., 2016, Gal et al., 2017]. Like classical active learning, deep active learning uses a variety of criteria for selecting data points. Uncertainty sampling is a major direction here as well [Gal et al., 2017, Yi et al., 2022, Ren et al., 2021, Zhan et al., 2022]. A well-known state-of-the-art method in this context is MC Dropout [Gal et al., 2017], which uses dropout at test time in order to create variants of an initially trained deep neural network model; averaging predictions over these model variants then yields estimates of class probabilities that can be used by standard acquisition functions.

One criticism of uncertainty sampling is that it tends to result in highly correlated queries [Sener and Savarese, 2017], which may prevent methods like MC Dropout from scaling well to large datasets. In fact, it was noted by Pinsler et al. [2019] that MC Dropout works very well on the MNIST dataset, but has a poor performance on the more complex CIFAR-10 data. This problem can be addressed with alternate selection principles or with the combination of uncertainty-based and alternate principles.

Several deep active learning methods do not focus on uncertainty. For example, a geometric point of view motivates the use of coresets in active sample selection. When optimizing an objective function on sets of points, a coreset of a set P of points is a subset of P whose optimal solution approximates the optimal solution for P. This principle of sparse approximation is deployed by [Sener and Savarese, 2017] for sample selection in active training of CNNs; specifically, they compute geometric center points for clusters of labeled points and then select unlabeled points whose distance to the nearest center is maximal. An interesting recent approach is Bayesian Batch [Pinsler et al., 2019], which also uses coresets for actively training CNNs, but in a Bayesian rather than a geometric setting.

Other methods use uncertainty in different ways in order to create more cost-efficient methods. One example of such an approach for training CNNs is the method used by [Wang et al., 2016], which assigns pseudo-labels for large batches of data points with high classification confidence. These pseudo-labels are used for re-training the model, which can result in a more effective process of selecting from the remaining unlabeled data points. Some methods combine self-supervised or semi-supervised methods with standard active learning methods. For instance, the method by [Yi et al., 2022] performs a pretext task (like rotation prediction) over the pool of unlabeled data points, which is then divided into batches based on the performance on the pretext task. Then, uncertainty-based active sampling is performed over one batch. By comparison, the method by [Siméoni et al., 2021] performs first an unsupervised feature learning task and then semi-supervised tasks in cycles.

In light of this related work, we propose a new deep active learning method that is mainly based on uncertainty sampling, yet combines uncertainty sampling with a diversity-based criterion in order to reduce the correlation of selected data points. The main novelty of our method is its way of estimating class probabilities: they are entirely derived from the probabilistic estimates produced by an SPN [Poon and Domingos, 2011] trained on the data representation extracted by the CNN model. The same SPN is used also for the diversity-based criterion by which we modify standard uncertainty sampling. SPNs are typically used for density estimation and have, to the best of our knowledge, not yet been applied to derive class probabilities for uncertainty sampling.

3 Preliminaries

In what follows, we assume an initial training dataset $\mathcal{D}_{train} = \{(x_i, y_i)\}_{i=0}^{N}$ of observations (x_i, y_i), where x_i is a data point and $y_i \in C = \{c_1, \ldots, c_{|C|}\}$ is the corresponding class label from a fixed set C of labels. Active learning means that, in addition, the learning algorithm has access to a pool $\mathcal{D}_{pool}$ of unlabeled data points. After processing the initial training dataset, the learning algorithm is permitted to select a small set of points from the pool, which will then be labeled by an oracle. The newly labeled set is added to the training set, and the model is retrained.

The goal of active learning is to train a high quality predictive model with less data than would be required for passively learning a model of similar predictive performance. A key aspect in achieving this goal is the mechanism for selecting points from the pool. A criterion for picking points is usually given in terms of an acquisition function $a(x, \mathcal{M})$, which takes as input the current model $\mathcal{M}$ and a point x from the pool, and assigns an informativeness value to x. In general, the acquisition strategy is to select a point in the pool that is maximally informative:

$$x^* = argmax_{x \in \mathcal{D}_{pool}} a(x, \mathcal{M}) \tag{1}$$

Acquisition functions can be based on various aspects, e.g., the distribution of the pool or the uncertainty with which the model makes predictions on individual pool data points. Here we focus on the latter type of acquisition function; in our experimental study, we work with two of the most popular uncertainty-based acquisition functions, namely Max Entropy and Variational Ratio.

Max Entropy Here uncertainty is expressed as predictive entropy [Shannon, 1948].

$$\mathbf{H}[y \mid x, \mathcal{D}_{train}] = -\sum_{c} p(y = c \mid x, \mathcal{D}_{train}) \log p(y = c \mid x, \mathcal{D}_{train}) \tag{2}$$

Variational Ratio The variational ratio [Freeman, 1965] provides a way to interpret certainty for data point x as the confidence value in predicting the most likely class y for x.

$$\text{var-ratio}[x] = 1 - \max_{y} p(y \mid x, \mathcal{D}_{train}) \tag{3}$$

The effectiveness of acquisition functions largely depends on how accurately the probability $p(y \mid x, \mathcal{D}_{train})$ can be approximated. Some common types of machine learning models, like logistic regression or deep neural networks, provide a probability value $p(y = c \mid x, \omega)$ (where ω refers to the model) as their output when making a prediction for x; this could be directly used as a surrogate for $p(y \mid x, \mathcal{D}_{train})$ in an acquisition function. While this method is simple and straightforward to implement, it may be highly susceptible to individual model parameters and thus not yield a good approximation of $p(y \mid x, \mathcal{D}_{train})$. In the neural network case, this problem has been addressed by the MC Dropout approach [Gal and Ghahramani, 2016] explained below.

Our main contribution is to provide and test an alternate approach to approximating $p(y \mid x, \mathcal{D}_{train})$ for deep active learning, based on SPNs. Before explaining our approach in detail, we briefly sketch the relevant background on MC Dropout and SPNs (where we focus on discriminative SPN models).

MC Dropout Dropout is a regularization technique in deep learning that is usually applied at training time [Srivastava et al., 2014]. [Gal and Ghahramani, 2016] showed that applying dropout many times in the test phase can help to estimate predictive uncertainty. Their method, called MC Dropout, uses dropout to create multiple versions ω_t, $t = 1, \ldots, T$ of the network model ω, and then estimates $p(y \mid x, \mathcal{D}_{train})$ by averaging the values of $p(y \mid x, \omega_t)$. This average is less reliant on individual parameters of ω than the probabilities $p(y = c \mid x, \omega)$ directly output by the network model ω when it makes predictions.

SPNs An SPN is a directed acyclic graph with layers of nodes of different types (leaf nodes, sum nodes, and product nodes) [Poon and Domingos, 2011]. Leaf nodes take as input values of some random variables $X^1, \ldots, X^d$; in our context, each random variable refers to one component of a d-dimensional data point x_i. The value of any non-leaf node is determined by the values of its children. The value of a sum node is a weighted sum of the values of its children, where the weights assigned to the children sum up to 1. The value of a product node is the product of its children's values. The value of the root can then be interpreted as the SPN's prediction of the probability with which data point x_i belongs to a specific class $c \in C$. (For classification problems with $|C|$ classes, one has to deploy $|C| - 1$ such DAGs.) Similar to a neural network, an SPN is determined by (a) its graphical structure, and (b) its parameters, which are the weights of the children of sum nodes. However, their interpretation is purely probabilistic.

Learning SPNs can be accomplished in various ways. Random SPN [Peharz et al., 2018] constructs a random region graph, which is then populated with tensors of SPN nodes. An alternate method is to construct an SPN based on convolutional product layers [Wolfshaar and Pronobis, 2020, Butz et al., 2019]. (In our experiments, we use the structure chosen by [Wolfshaar and Pronobis, 2020].) In order to learn the weights, standard optimization procedures for minimizing a loss function given the training data are used, see, e.g., [Peharz et al., 2018].

4 SPN-Based Uncertainty Sampling for Active Learning of CNNs

SPNs are deep probabilistic models, which are mostly used for the purpose of density estimation. Therefore, they are an apt model for extracting the distribution of a dataset. The main idea of our deep active learning approach is to use SPNs for extracting probability estimates that can be used in standard uncertainty-based acquisition functions when actively learning a deep neural network classifier. To this end, we introduce two novel techniques for applying SPNs.

4.1 Training an SPN over the Representation of Data

Typically, an SPN is trained directly on the observed data. However, if one has trained a different type of model (e.g., a deep neural network) on the observed data, and extracted a new feature representation of the data, one can train an SPN on this representation rather than on the original data. If the observed data is complex, or contains redundancies or noise, then the representation extracted from the data is expected to better capture the distribution of the data. Thus, if an SPN is trained in order to capture the true data distribution, it might be more useful to train it on previously extracted features than to train it on the original data; this is the first technique we propose in this paper. Formally, instead of approximating $p(y = c \mid x, \mathcal{D}_{train})$, we use an SPN to approximate $p(y = c \mid z, \mathcal{D}_{train})$, where z is the representation of x. In practice, we take the output of the final dense layer (before the output layer) of a trained CNN and consider it as a feature representation of the input data, on which the SPN

is trained. The probability estimates computed by the SPN can then be used as $p(y = c \mid x, \mathcal{D}_{train})$ values in acquisition functions like max entropy or variational ratio.

4.2 Weighting Examples Based on Disaggregation

The second technique we propose is to use the classification output of the SPN in order to calculate weights by which to adjust standard acquisition functions and make them less prone to select highly correlated data points. Intuitively, as will become evident below, our method gives more weight to the diversity of predicted classes when determining uncertainty.

Suppose a trained SPN model is used as a classifier on a data point x multiple times, each time independently applying dropout at classification time. If this results in the same classification for x every time, one could consider x as a *strongly classified* point. By contrast, x is *weakly classified* by the SPN model if there are many different classes predicted for x among the multiple predictions obtained. How weakly x is classified can be interpreted as an additional indicator – on top of the probability output by the SPN – of the level of uncertainty with which the SPN makes a prediction on x. The approach of our second new technique is to modify standard acquisition functions – like variational ratio or max entropy – by a weight factor based on this interpretation of SPN model uncertainty.

An SPN model gives rise to T different SPN models SPN_i, $1 \leq i \leq T$, by T independent experiments of applying dropout to the original SPN. Now let $\mathrm{SPN}_i(x)$ denote the class label predicted by SPN_i on data point x, and let C be the set of class labels. We then define

$$predictionset(x) = \{\mathrm{SPN}_i(x) \mid 1 \leq i \leq T\} \tag{4}$$

Then, the weight for point x is obtained from

$$weight(x) = \frac{|predictionset(x)|}{|C|} \tag{5}$$

Finally, the chosen uncertainty-based acquisition function a (such as variational ratio or max entropy) is multiplied with the weight function, which then results in the following acquisiton strategy.

$$x^* = argmax_{x \in \mathcal{D}_{pool}} weight(x) a(x, \mathcal{M}) \tag{6}$$

4.3 SPN-CNN – A New Deep Active Learning Method

The two SPN-based techniques that we introduced above are combined in our proposed model architecture for deep active learning, called SPN-CNN. As depicted in Figure 1, the three main components of our architecture are (i) a CNN, (ii) an SPN, and (iii) an uncertainty-based acquisition function (like variational ratio or max entropy).

The training process works as follows.

1. An initial CCN model is trained on a given set of initial training data points.
2. An initial SPN is trained on the output of the final dense layer of the obtained CNN. Specifically, for each initial training data point x_0, the output of the dense layer is fed to the SPN, together with the class label of x_0. The original data x_0 itself is not fed to the SPN.
3. The SPN is used to make predictions on pool points, which allows us to calculate (i) estimates of $p(y = c \mid x, \mathcal{D}_{train})$ values as needed in uncertainty-based acquisition functions, and (ii) weights as proposed in Section 4.2. The weight for each point x from the data pool is calculated by (5) and multiplied by the output of the acquisition function ($weight(x)a(x, \mathcal{M})$).
4. A new data point x' is selected following our acquisition strategy in (6), and the class label y' of x' is obtained from the oracle. (This step may be repeated a predefined number of times to collect a batch of new labeled data points.)
5. The CNN block is retrained using the (batch of) newly acquired training data (x', y'); the corresponding outputs of the dense layer are used to retrain the SPN.
6. If the sample budget is used up, then stop, else go to Step 3.

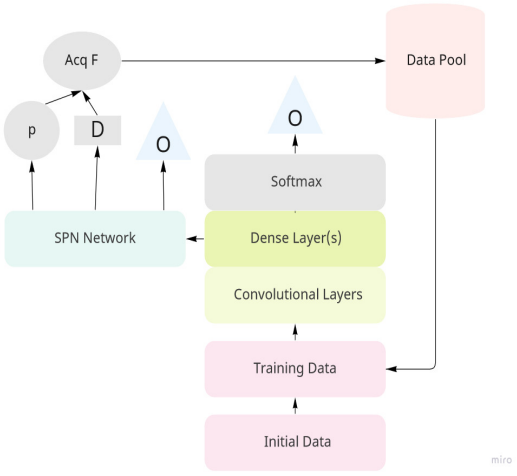


Figure 1: SPN-CNN deep active learning approach.

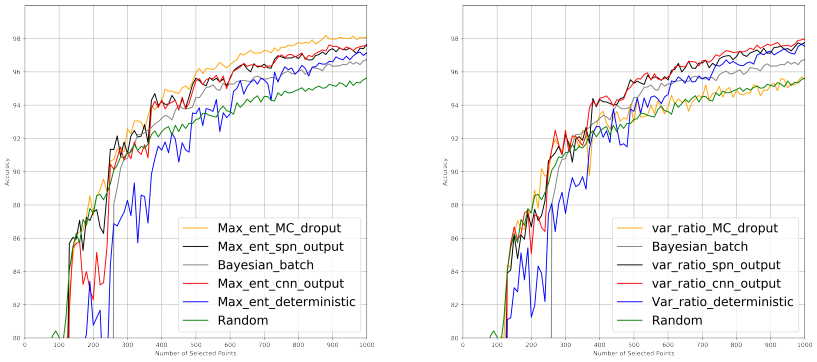
Note that the output of the SPN (obtained from applying $\mathrm{softmax}$ to the vector of prediction values of the individual DAGs; depicted as "O" in Figure 1) is needed when training the SPN, but it is not used by the acquisition function itself. (However, it is used in selecting data points, since the weights by which the acquisition function are multiplied are calculated based on SPN outputs.)

Our model has two classifiers: a CNN trained over observed data, and an SPN trained over the CNN's representation of data. Both of them can be used for predicting class labels, as indicated by the two triangles labeled "O" in Figure 1. When evaluating our approach in Section 5, we consider both options separately, one in which the final prediction is the output of the CNN and one in which the final prediction is the output of the SPN.

5 Experimental Analysis

We tested SPN-CNN (both with final predictions made by the CNN and with final predictions made by the SPN) on three standard benchmark datasets: MNIST [Deng, 2012], CIFAR-10 [Krizhevsky et al., accessed May 2022], and Fashion-MNIST [Xiao et al., 2017]. We compared our method to four others, which differ in the way data points are actively sampled. Two simple baselines train the same initial CNN as our method but actively sample as follows: (i) the method *Random* samples randomly without replacement from the pool, where each pool point has the same probability of being sampled; (ii) the method *Deterministic* uses standard acquisition functions, where class probabilities are simply the outputs of the CNN before applying softmax. Two state-of-the-art competitors that we tested are (iii) MC Dropout [Gal et al., 2017] and (iv) Bayesian Batch [Pinsler et al., 2019]; this way, we compare SPN-CNN to another uncertainty sampling method as well as to a method using an entirely different approach. Note that SPN-CNN, as well as *Deterministic* and MC Dropout, compute class probabilities to be used in acquisition functions; we tested both Max Entropy and Variational Ratio as acquisition functions. In all our experiments, we used the same SPN structure, but a different CNN structure for each dataset. Details on the experimental setup, such as the SPN and CNN architectures, hyperparameters, off-the-shelf tools used, details on dropout procedures, etc., are provided in the appendix.

How to read our plots All our result plots have the number of training data points (initial training set size plus actively sampled points) on the x-axis and accuracy on the y-axis. The y-intercept of a curve equals the accuracy of the corresponding model when trained only on the initial training data; each subsequent point represents the accuracy after processing one additional batch of acquired points. Each curve is the average over multiple runs, with randomly chosen training, validation, and pool sets per run and per method. Note that the curves for SPN-CNN with SPN output and for Bayesian Batch often have substantially different y-intercepts than the other methods, since they



(a) Max Entropy (b) Variational Ratio

Figure 2: Accuracy results on the MNIST data, averaged over five runs (std dev is given in Appendix B).

correspond to outputs from other model structures; the remaining four methods produce outputs based on the same CNN structure and thus have similar y-intercepts.

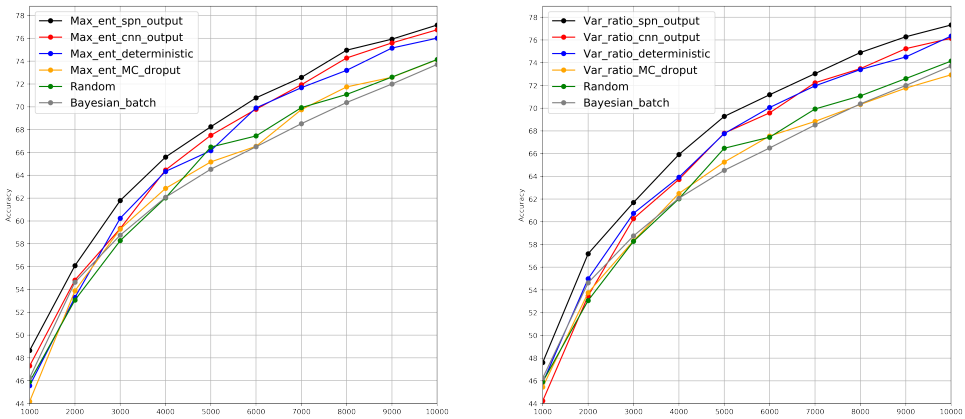
MNIST The CNN structure used is similar to LeNet [LeCun et al., 1998]; see Appendix A. All competing methods were given the same test data, which is the predefined test set consisting of $10K$ data points. For each run and each method, we sample other data sets from the predefined $50K$ training data points as follows: the $50K$ points are shuffled at random, $40K$ points are randomly chosen for the pool, $5K$ points for the validation set, and from the remaining $5K$ points we randomly choose an initial training set of only 20 points. Acquisition was performed 98 times, each time picking 10 points (for a total of $1K$ training points in the end), following the respective acquisition strategy and then updating the model. Figure 6 shows the resulting accuracy, averaged over five runs. The same figure with standard deviation added is shown in Appendix B.

In both cases, SPN-CNN (both with SPN output and with CNN output) performs better than the two baselines and than Bayesian Batch. After five batch acquisitions with Max Entropy, MC Dropout surpasses SPN-CNN, but it is as bad as the *Random* baseline when applied in Variational Ratio. Note that, in the first few acquisitions, *Random* performs as well as the other methods (even better than *Deterministic*), but it cannot keep up later on. One possible explanation is that in the first steps active learning gains little to no advantage over passive learning, but when the model becomes more refined, active sample selection makes a bigger difference.

This experiment also demonstrates how strong the performance of an SPN can be when trained over the feature representation of data. In the existing literature, the highest reported accuracy of an SPN model, trained on the full predefined MNIST training set of size $50K$, is around 98% [Peharz et al., 2018, Wolfshaar and Pronobis, 2020]. By comparison, SPN-CNN with SPN output obtained an accuracy of 97.6% (std deviation 0.045%), trained on only 1000 data points. Hence, training SPNs on the representation of data appears promising.

CIFAR-10 CIFAR-10 is a much more complex dataset than MNIST, and thus requires a more complex CNN structure (we used VGG [Simonyan and Zisserman, 2014]) and larger training set sizes than MNIST. The predefined test set of $10K$ points was used in all evaluations. We performed two tests. The first one is similar to the setups used by Siméoni et al. [2021] and Yi et al. [2022], with $1K$ initial training data points and nine acquisitions in batches of size $1K$. The second one (performed only for Max Entropy) used $5K$ initial training data points and four acquisition batches of 5K points each. In both cases and for each method tested, we shuffle the predefined training set of $40K$ points, then randomly choose a pool of size $30K$ and a validation set of size $5K$; the initial training set is sampled at random from the remaining $5K$ points.

The result for the first setting is in Figure 3. Here SPN-CNN with SPN output is the winner, followed by a tie between SPN-CNN with CNN output and *Determinisitic*. Neither MC Dropout nor Bayesian Batch can compete; they behave similarly to *Random*. Figure 4 shows the result for the second setting, with larger initial training data sets and larger step sizes. Here, MC Dropout and *Deterministic* are



(a) Max Entropy (b) Variational Ratio

Figure 3: Accuracy results on the CIFAR-10 data, averaged over five runs (std dev is given in Appendix B). Initial training set size and acquisition batch size were 1000.

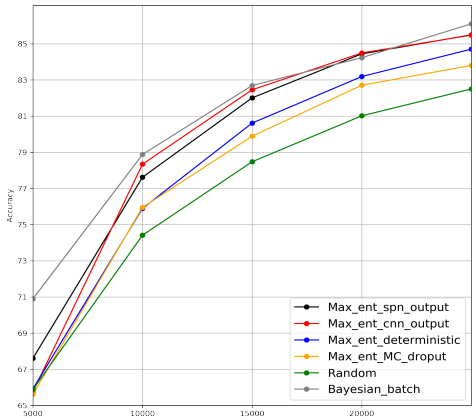
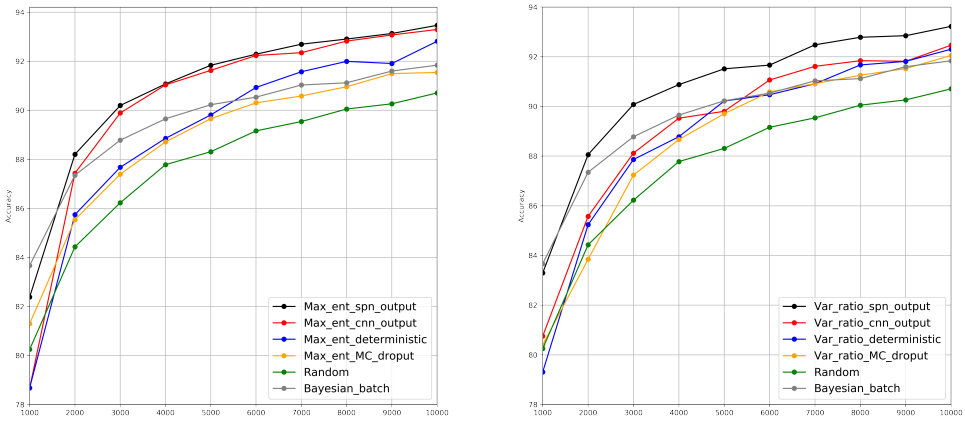


Figure 4: As in Figure 3, but with initial training set size and acquisition batch size of 5000.

better than *Random*, but the winners are the two SPN-CNN methods, jointly with Bayesian Batch. Interestingly, Bayesian Batch needs the larger initial training data set/batch sizes to be competitive with SPN-CNN. This suggests that SPN-CNN is particularly effective when data is less abundant.

Fashion-MNIST Here, we use VGG as the CNN structure. The predefined test set has $10K$ points; from the $50K$ given training data points, we randomly sample $40K$ for the pool, $5K$ for validation, and then $1K$ from the remaining ones as initial training data. We performed 9 acquisitions of $1K$ points each. Figure 9 shows the results averaged over three runs. SPN-CNN with SPN output outperforms all other methods for Variational Ratio. For Max Entropy, both variants of SPN-CNN are the winners. For both acquisition functions, Bayesian Batch is still doing fairly well after the first batch acquisition, but, like MC Dropout, cannot compete with SPN-CNN. In fact, Bayesian Batch and MC Dropout here seem no better than the *Deterministic* baseline.

Run-time comparison For a run-time comparison, we used a local ubuntu Server 20.04.1 with 6-core, 64GB RAM, 128 GB SSD disk drive, GPU NVIDIA Corporation GP100GL [Tesla P100 PCIe 16GB]. Table 1 shows the resulting run-times in seconds. For MNIST, we acquired 98 batches of 10 points each plus the training time for initial set with 20 points; for both CIFAR-10 and Fashion-MNIST, we report the time needed for the acquisition of 9 batches of 1000 points each plus the training time for initial set with 1000 points. In all run-time experiments, we trained the SPN in 650 epochs. Not surprisingly, random acquisition requires the least time. Overall, SPN-CNN's run-time is up to 2.5 times that of MC Dropout, yet only around 30%–50% of that for Bayesian Batch.



(a) Max Entropy (b) Variational Ratio

Figure 5: Accuracy results on the Fashion-MNIST data, averaged over three runs (std dev is given in Appendix B).

Dataset	Random	Deterministic Max	Deterministic Var	Bayesian Batch	MC Dropout Max	MC Dropout Var	SPN-CNN Max	SPN-CNN Var
MNIST	1,527	1,633	1,617	10,603	2,111	2,277	5,321	5,288
CIFAR-10	4,072	4,058	4,062	16,709	5,800	6,231	6,539	6,404
Fashion-MNIST	3,920	3,952	3,984	32,447	6,254	6,997	10,241	10,242

Table 1: Run-time (in seconds). "Max" is for Max Entropy, "Var" for Variational Ratio.

6 Conclusions

We proposed SPN-CNN, a new method for active learning with CNNs, which estimates class probabilities with an SPN trained over the representation extracted by the CNN. It also uses the SPN in order to somewhat balance uncertainty with disaggregation in active sampling. In our experimental study, in terms of accuracy, SPN-CNN with SPN output is not outperformed by any of the tested competitors, with two exceptions: (i) MC Dropout outperforms SPN-CNN on MNIST when using Max Entropy, and (ii) Bayesian Batch and SPN-CNN with CNN output, while mostly worse than SPN-CNN with SPN output, perform slightly better or comparable for the large training data set and batch sizes we tested on CIFAR-10; large initial training sets however are a less likely situation in the context of active learning. Our study confirms previous reports that MC Dropout is not very effective on more complex data (like CIFAR-10 and Fashion-MNIST). Apparently, Bayesian Batch requires large initial training sets to compete with SPN-CNN. In the sum, our method to estimate class probabilities with an SPN trained on a CNN's feature representation appears to be very effective, at a reasonable run-time cost compared to state-of-the-art methods.

Limitations Our method has potential limitations. We tested it only on image data with CNNs; it is unclear whether the same approach will work well combined in other applications of deep learning and with other network architectures, e.g., autoencoders. Our CIFAR-10 experiment also shows that the effectiveness of the tested methods depends on the size of the initial training dataset and/or the acquisition batch size. More experiments are needed to determine trends in this regard.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), through the Discovery Grants program and the Canada Research Chairs program. It was also supported through the Canada CIFAR AI Chairs program; S. Zilles holds an Affiliate Chair with the Alberta Machine Intelligence Institute (Amii).

References

K. Brinker. Incorporating diversity in active learning with support vector machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 59–66, 2003.

J. Brownlee. How to develop a cnn from scratch for cifar-10 photo classification. `https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/`, 2019. Accessed: 2013-04-13.

C. J. Butz, J. S. Oliveira, A. E. dos Santos, and A. L. Teixeira. Deep convolutional sum-product networks. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 3248–3255, 2019.

L. Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

L. C. Freeman. *Elementary Applied Statistics: For Students in Behavioral Science*. New York: Wiley, 1965.

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the International Conference on Machine Learning*, pages 1050–1059, 2016.

Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1183–1192, 2017.

A. Krizhevsky, V. Nair, and G. Hinton. CIFAR-10. *Online Repository*, accessed May 2022. URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the Annual Conference of the ACM Special Interest Group in Information Retrieval (SIGIR)*, pages 3–12, 1994.

V.-L. Nguyen, M. H. Shaker, and E. Hüllermeier. How to measure uncertainty in uncertainty sampling for active learning. *Machine Learning*, 111(1):89–122, 2022.

R. Peharz, A. Vergari, K. Stelzner, A. Molina, M. Trapp, K. Kersting, and Z. Ghahramani. Probabilistic deep learning using random sum-product networks. *arXiv preprint arXiv:1806.01910*, 2018.

R. Pinsler, J. Gordon, E. Nalisnick, and J. M. Hernández-Lobato. Bayesian batch active learning as sparse subset approximation. *Advances in Neural Information Processing Systems*, 32, 2019.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690, 2011.

P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang. A survey of deep active learning. *ACM Computing Surveys (CSUR)*, 54(9):1–40, 2021.

O. Sener and S. Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

O. Siméoni, M. Budnik, Y. Avrithis, and G. Gravier. Rethinking deep active learning: Using unlabeled data at model training. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 1220–1227, 2021.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

S. Sinha, S. Ebrahimi, and T. Darrell. Variational adversarial active learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5972–5981, 2019.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

D. Wang and Y. Shang. A new active labeling method for deep learning. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 112–119, 2014.

K. Wang, D. Zhang, Y. Li, R. Zhang, and L. Lin. Cost-effective active learning for deep image classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(12):2591–2600, 2016.

J. Wolfshaar and A. Pronobis. Deep generalized convolutional sum-product networks. In *Proceedings of the International Conference on Probabilistic Graphical Models (PGM)*, pages 533–544, 2020.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv*, cs.LG/1708.07747, 2017.

J. S. K. Yi, M. Seo, J. Park, and D.-G. Choi. Using self-supervised pretext tasks for active learning. *arXiv preprint arXiv:2201.07459*, 2022.

X. Zhan, Q. Wang, K.-h. Huang, H. Xiong, D. Dou, and A. B. Chan. A comparative survey of deep active learning. *arXiv preprint arXiv:2203.13450*, 2022.

A Details on experimental setup

This section describes the deep SPN and CNN structures as used in our experiments, including hyperparameter settings etc.

A.1 SPN

For implementing the SPN, we use the lib-spn keras library which is inherited from keras.

The SPN structure used in all our experiments is Deep Convolutional SPN [Wolfshaar and Pronobis, 2020], consisting of a leaf layer (with 16 components for each variable) to which we apply dropout (with dropout rate of .3 for FMNIST and CIFAR-10, and dropout rate of .05 for MNIST), then a product layer, sum layer (with 16 sum nodes), product layer, sum layer (with 32 sum nodes), product layer, sum layer (with 32 sum nodes), product layer, sum layer (with 64 sum nodes), product layer, sum layer (with 64 sum nodes), root layers (one for each class). Since the values returned by the root layers were very small, we additionally applied softmax at the last step.

We always trained our SPNs over 650 epochs, using the Adam optimizer, and with sparse categorical entropy as loss function. The learning rate was manually tuned to .08.

We first set the learning rate by trying out values from .005 to .1 in steps of size .01 with a fixed dropout rate of .1. We chose the learning rate that maximized average accuracy on the validation set over two runs. Next, we kept the learning rate fixed at .08, and then tested values from .05 to .5 in steps of size .05 to determine a good dropout rate. Again, we chose the value that maximized average accuracy on the validation set over two runs. Moreover, we tried dropout in layers other than the leaf layer but this always decreased the accuracy.

A.2 CNN for MNIST

For all methods except Bayesian Batch, we use a structure similar to LeNet [LeCun et al., 1998]: two convolutional layers (filter size 32), maxpool layer, two convolutional layers (filter size 64), dropout layer (dropout rate .5), flatten layer, dense layer (128 neurons), dropout layer (dropout rate .5), dense layer with softmax activation. The output of the dense layer is reshaped (16×8) to use as input for the SPN. We used the SGD optimizer with a learning rate of .001 and as loss function we chose categorical cross entropy. The number of epochs was set to 100, with a batch size of 120.

We set hyperparameters by assessing accuracy on the validation set for several values of the hyperparameters. For the learning rate we tried .001 to .1 with step size of .001, each with a dropout rate of .25 and a dropout rate of .5. After fixing the learning rate and the dropout rate, we tested three different numbers of neurons for the dense layer (128, 256, and 512) but since this did not seem to affect the accuracy, we chose the smallest of the tested sizes.

For Bayesian Batch, we directly used the authors' code [Pinsler et al., 2019], since their architecture is totally different from the ones used by the other tested methods.

A.3 CNN for CIFAR-10

For all methods except Bayesian Batch, the CNN structure used for our CIFAR-10 experiments is VGG [Simonyan and Zisserman, 2014], and consists of a convolutional layer (filter size 64), batchNormalization layer, convolutional layer (filter size 64), batchNormalization layer, maxpool layer, dropout layer (dropout rate .2), convolutional layer (filter size 128), batchNormalization layer, convolutional layer (filter size 128), batchNormalization layer, maxpool layer, dropout layer (dropout rate .3), convolutional layer (filter size 256), batchNormalization layer, convolutional layer (filter size 256), batchNormalization layer, convolutional layer (filter size 256), batchNormalization layer, maxpool layer, dropout layer (dropout rate .4), convolutional layer (filter size 512), batchNormalization layer, convolutional layer (filter size 512), batchNormalization layer, convolutional layer (filter size 512), batchNormalization layer, maxpool layer, dropout layer (dropout rate .5), flatten layer, dense layer (with 512 neurons), batchNormalization layer, dropout layer (dropout rate .5), softmax layer with 10 neurons.

The output of the dense layer was reshaped (16×32) as input for the SPN. We used SGD with learning rate .001 as optimizer and categorical cross entropy as loss function. The number of epochs was set to 250, with a batch size of 100.

We took the VGG model directly from [Brownlee, 2019]. Since the structure was established, we did not attempt hyperparameter tuning as much as with the model used for MNIST. We only tried the learning rates .01 and .001, as well as the dropout rates .25, .3 and .5 for the two last dropout layers. For the number of neurons in the dense layer we tried 256 and 512.

For Bayesian Batch, we directly used the authors' code [Pinsler et al., 2019], since their architecture is totally different from the ones used by the other tested methods.

A.4 CNN for Fashion-MNIST

For all methods apart from Bayesian Batch, the structure and hyperparameter settings were the same as for the CIFAR-10 experiments, except that the learning rate was .0005.

We did not attempt to tune the dropout rate at all (keeping it the same as for the CIFAR-10 data), and tried only three different values for the learning rate (.1, .001, and .0005), picking .0005 as it performed best on the validation set.

For Bayesian Batch, we directly used the authors' code [Pinsler et al., 2019], since their architecture is totally different from the ones used by the other tested methods.

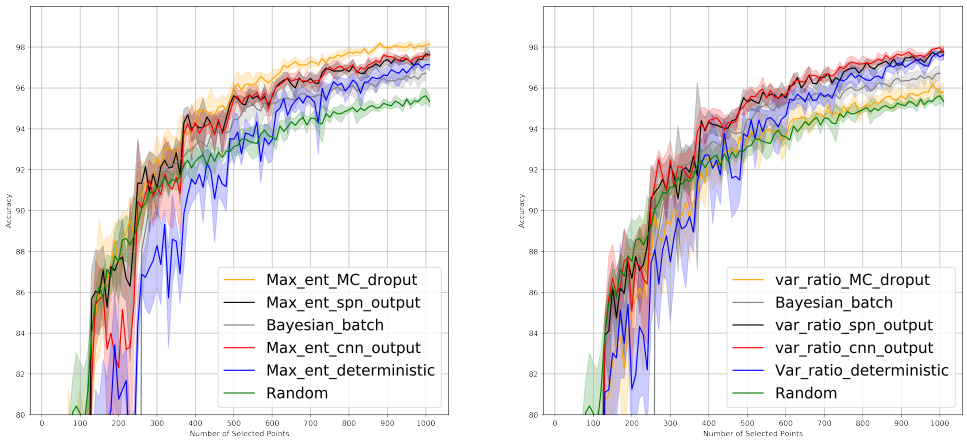
B Result plots with standard deviation

This section presents the same plots as in Section 5, but with standard deviation indicated in addition. For the sake of readability, we did not show standard deviation in the plots in Section 5. As one can see here, Figures 6, 7, and 9 are not as easy to analyze as the corresponding figures in the main paper. Therefore, for each of these three figures, we here added a table showing the numerical values of accuracy and standard deviation plotted in the figures. For MNIST, Figure 6 plots accuracy at 100 different x-values; here we chose 10 equidistant values for the corresponding table (Table 2). Note that, in accordance with the results reported in the main paper, the standard deviation was calculated over five runs for MNIST and CIFAR-10, and over three runs for Fashion-MNIST.

C Code

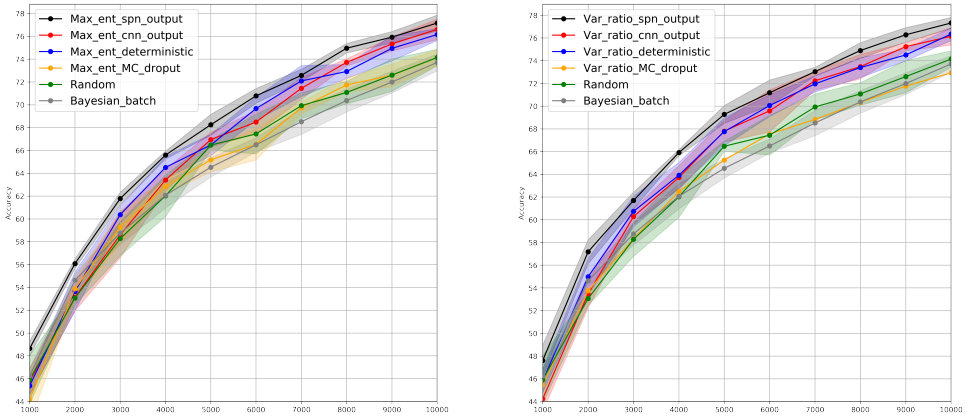
Code that we used off the web is the following

- For MC Dropout we used the code from [Gal et al., 2017]. We modified the CNN in their code to be the same as the CNN that we use in SPN-CNN, *Random* and *Deterministic*.
- For Bayesian Batch, we used the code from [Pinsler et al., 2019].
- In our code we used part of the code in [Gal et al., 2017] for preparing the initial set and new training set in each acquisition.



(a) Max Entropy (b) Variational Ratio

Figure 6: Accuracy results and error bars on the MNIST data, averaged over five runs, with std dev indicated.



(a) Max Entropy (b) Variational Ratio

Figure 7: Accuracy results and error bars on the CIFAR-10 data, averaged over five runs, with std dev indicated.

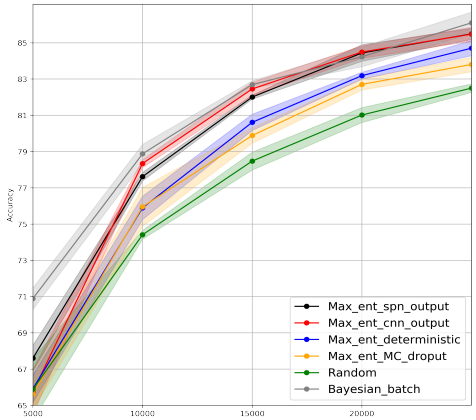


Figure 8: As in Figure 7, but with initial training set size and acquisition batch size of 5000.

Method	100	200	300	400	500
SPN-CNN (SPN, max-ent)	74.29 ± 1.65	86.67 ± .98	92.23 ± 1.46	94.34 ± 0.63	95.31 ± 0.39
SPN-CNN (CNN, max-ent)	62.62 ± 4.68	82.30 ± 4.90	91.16 ± 1.29	94.31 ± 0.37	95.46 ± 0.74
Deterministic (max-ent)	60.25± 5.09	80.77± 3.08	88.28± 2.06	91.29± 0.64	93.49± 1.28
MC Dropout (max-ent)	76.62± 4.77	87.37± 1.93	92.58± 1.12	94.2± 1.12	95.94± 0.50
SPN-CNN (SPN, var-ra)	74.29± 3.64	86.67± 1.89	92.23± 0.53	94.34± 0.57	95.31± 0.49
SPN-CNN (CNN, var-ra)	58.98± 9.66	85.07± 3.01	92.49± 0.98	94.41± 0.42	95.37± 0.63
Deterministic (var-ra)	63.64± 6.80	81.25± 2.99	87.49± 3.00	92.7± 0.96	93.62± 0.44
MC Dropout (var-ra)	74.4±2.97	85.26±2.07	89.31±2.22	92.68± 0.71	93.43± 0.36
Bayesian Batch	9.86±0.69	10.20±0.73	90.79±1.27	93.19±0.76	94.46±0.65
Random	80.06±2.47	87.5± 2.06	91.14±0.39	92.4±0.83	93.13± 0.41
Method	600	700	800	900	1000
SPN-CNN (SPN,max-ent)	95.51 ± 0.27	96.524 ± 0.34	96.92 ± 0.34	97.13 ± 0.26	97.77 ± 0.32
SPN-CNN (CNN, max-ent)	95.67± 0.64	96.24 ± 0.45	97.00 ± 0.22	97.53 ± 0.28	97.55 ± 0.18
Deterministic (max-ent)	93.47± 1.05	95.50± 0.29	96.07± 0.45	96.61± 0.43	97.15 ± 0.18
MC Dropout (max-ent)	96.42± 0.44	97.30± 0.25	97.77± 0.27	97.97± 0.13	98.09± 0.16
SPN-CNN (SPN, var-ra)	95.51± 0.31	96.52± 0.34	96.92± 0.24	97.13± 0.18	97.77± 0.06
SPN-CNN (CNN, var-ra)	95.68± 0.40	96.67± 0.31	97.18± 0.39	97.59± 0.19	97.98± 0.05
Deterministic (var-ra)	94.63± 0.98	95.57±0.53	96.48±0.15	97.24±0.27	97.52±0.2
MC Dropout (var-ra)	93.59±0.71	94.57±0.24	95.03±0.27	95.64± 0.40	95.75±0.29
Bayesian Batch	95.28±0.61	95.57±0.49	96.03±0.26	96.33±0.16	96.73±0.35
Random	93.62± 0.28	94.52±0.15	94.93±0.13	95.17± 0.18	95.62±0.36

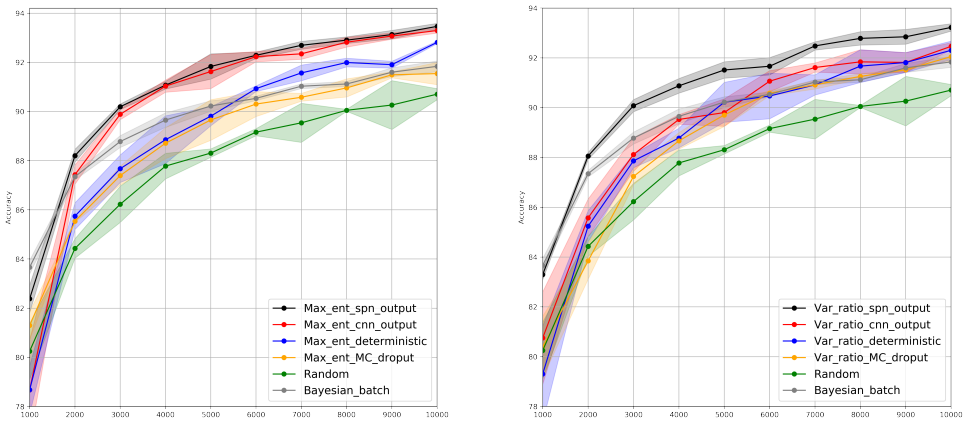
Table 2: Accuracy and standard deviation for MNIST over five runs, for 10 selected (equidistant) acquisition steps. This table corresponds to Figure 6.

Method	1000	2000	3000	4000	5000
SPN-CNN (SPN, max-ent)	48.65 ± 0.66	56.09 ± 0.42	61.79 ± 0.56	65.59 ± 0.35	68.25 ± 0.92
SPN-CNN (CNN, max-ent)	45.77 ± 1.16	53.13 ± 1.18	58.61 ± 1.99	63.41 ± 0.95	66.95 ± 0.63
Deterministic (max-ent)	45.39± 0.83	53.65± 1.71	60.37± 0.74	64.52± 0.97	66.48± 0.94
MC Dropout (max-ent)	44.19± 2.03	53.87± .9	59.28± 1.41	62.85± 0.65	65.17± 1.04
SPN-CNN (SPN, var-ra)	47.62± 1.36	57.18± 1.06	61.69± 0.73	65.92± 0.27	69.27± 0.79
SPN-CNN (CNN, var-ra)	44.23± 1.07	53.48± 1.23	60.28± 0.68	63.73±1.03	67.8± 0.96
Deterministic (var-ra)	45.84± 1.01	54.98± 1.61	60.74± 1.35	63.92± 1.13	67.76± 1.23
MC Dropout (var-ra)	45.45±1.31	53.75±0.84	58.33±0.57	62.49± 0.55	65.26± 1.6
Bayesian Batch	46.25±0.65	54.63±0.5	58.76±0.98	62.07±1.11	64.53±0.84
Random	45.92±2.1	53.07± 0.6	58.28±1.5	62.02±1.8	66.47± 0.45
Method	6000	7000	8000	9000	10000
SPN-CNN (SPN,max-ent)	70.79 ± 0.65	72.58 ± 0.54	74.96 ± 0.42	75.92 ± 0.49	77.17 ± 0.69
SPN-CNN (CNN, max-ent)	68.49± 1.02	71.44 ± 0.97	73.72 ± 0.31	75.33 ± 0.62	76.61 ± 0.9
Deterministic (max-ent)	69.67± 1.02	72.09± 1.35	72.92± 0.62	74.95± 1.1	76.16 ± 0.52
MC Dropout (max-ent)	66.54± 1.38	69.75± 0.54	71.74± 0.62	72.59± 0.99	74.13± .7
SPN-CNN (SPN, var-ra)	71.18± 1.12	73.04± 0.38	74.9± 0.7	76.28± 0.64	77.32± 0.48
SPN-CNN (CNN, var-ra)	69.59± 1.85	72.24± 0.81	73.48± 1.09	75.24± 0.36	76.15± 0.77
Deterministic (var-ra)	70.06± 0.96	71.96±0.72	73.4±0.87	74.5±0.53	76.34±0.55
MC Dropout (var-ra)	67.55±1.03	68.83±0.7	70.33±0.84	71.76± 0.45	72.94±1.02
Bayesian Batch	66.5±0.62	68.5±1.11	70.38±1.00	71.99±0.96	73.71±0.73
Random	67.45± 1.74	69.92±1.13	71.08±.99	72.6± 1.42	74.15±0.72

Table 3: Accuracy and standard deviation for CIFAR-10 over five runs, after processing 1000, 2000, . . . , 10,000 data points (i.e., for the initial training data set and the subsequent 9 batch acquisitions). This table corresponds to Figure 7.

Method	1000	2000	3000	4000	5000
SPN-CNN (SPN, max-ent)	82.38 ± 0.56	88.2 ± 0.26	90.2 ± 0.11	91.07 ± 0.16	91.83 ± 0.51
SPN-CNN (CNN, max-ent)	78.67 ± 2.05	87.43 ± 0.39	89.89 ± 0.22	91.03 ± 0.25	91.62 ± 0.69
Deterministic (max-ent)	78.68± 0.98	85.74± 0.54	87.67± 0.56	88.85± 0.97	89.81± 0.39
MC Dropout (max-ent)	81.29± 0.56	85.54± .04	87.39± 0.39	88.71± 0.63	89.66± 0.83
SPN-CNN (SPN, var-ra)	83.3± 0.2	88.06± 0.14	90.08± 0.25	90.87± 0.29	91.51± 0.33
SPN-CNN (CNN, var-ra)	80.75± 1.85	85.58± 0.75	88.12± 0.68	89.53± 0.19	89.80± 0.55
Deterministic (var-ra)	79.03± 1.92	85.24± 0.63	87.86± 0.27	88.78± 0.39	90.22± 0.8
MC Dropout (var-ra)	80.40±1.23	83.85±0.73	87.24±0.32	88.67± 0.33	89.71± 0.46
Bayesian Batch	83.67±0.28	87.35±0.13	88.78±0.24	89.65±0.30	90.23±0.21
Random	80.25±1.14	84.43± 0.40	86.23±0.74	87.87±0.52	88.31± 0.17
Method	6000	7000	8000	9000	10000
SPN-CNN (SPN,max-ent)	92.27 ± 0.13	92.7 ± 0.16	92.9 ± 0.13	93.13 ± 0.17	93.46 ± 0.12
SPN-CNN (CNN, max-ent)	92.23± 0.2	92.35 ± 0.22	92.81 ± 0.2	93.07 ± 0.12	93.29 ± 0.1
Deterministic (max-ent)	90.93± 0.11	91.57± 0.32	92± 0.18	91.9± 0.14	92.81 ± 0.05
MC Dropout (max-ent)	90.3± 0.51	90.57± 0.16	90.96± 0.36	91.5± 0.12	91.54± 0.45
SPN-CNN (SPN, var-ra)	91.66± 0.35	92.48± 0.17	92.78± 0.27	92.84± 0.30	93.22± 0.15
SPN-CNN (CNN, var-ra)	91.06± 0.42	91.61± 0.18	91.84± 0.49	91.82± 0.40	92.46± 0.15
Deterministic (var-ra)	90.47± 0.91	90.91±0.40	91.67±0.66	91.81±0.40	92.3±0.37
MC Dropout (var-ra)	90.58±0.1	90.92±0.18	91.26±0.11	91.53± 0.03	92.04±0.18
Bayesian Batch	90.54±0.1	91.03±0.1	91.12±0.10	91.60±0.17	91.84±0.23
Random	89.16± 0.14	89.54±0.80	90.05±0.04	90.26± 0.99	90.71±0.23

Table 4: Accuracy and standard deviation for Fashion-MNIST over three runs, after processing 1000, 2000, . . . , 10,000 data points (i.e., for the initial training data set and the subsequent 9 batch acquisitions). This table corresponds to Figure 9.



(a) Max Entropy (b) Variational Ratio

Figure 9: Accuracy results and error bars on the Fashion-MNIST data, averaged over five runs, with std dev indicated.

Supplementary Material for Submission 2473

Anonymous Author(s)
Affiliation
Address
email

Abstract

This document contains the supplementary information for Submission 2473.

1 Details on experimental setup

This section describes the deep SPN and CNN structures as used in our experiments, including hyperparameter settings etc.

1.1 SPN

For implementing the SPN, we use the lib-spn keras library which is inherited from keras.

The SPN structure used in all our experiments is Deep Convolutional SPN [**?**], consisting of a leaf layer (with 16 components for each variable) to which we apply dropout (with dropout rate of .3 for FMNIST and CIFAR-10, and dropout rate of .05 for MNIST), then a product layer, sum layer (with 16 sum nodes), product layer, sum layer (with 32 sum nodes), product layer, sum layer (with 32 sum nodes), product layer, sum layer (with 64 sum nodes), product layer, sum layer (with 64 sum nodes), root layers (one for each class). Since the values returned by the root layers were very small, we additionally applied softmax at the last step.

We always trained our SPNs over 650 epochs, using the Adam optimizer, and with sparse categorical entropy as loss function. The learning rate was manually tuned to .08.

We first set the learning rate by trying out values from .005 to .1 in steps of size .01 with a fixed dropout rate of .1. We chose the learning rate that maximized average accuracy on the validation set over two runs. Next, we kept the learning rate fixed at .08, and then tested values from .05 to .5 in steps of size .05 to determine a good dropout rate. Again, we chose the value that maximized average accuracy on the validation set over two runs. Moreover, we tried dropout in layers other than the leaf layer but this always decreased the accuracy.

1.2 CNN for MNIST

For all methods except Bayesian Batch, we use a structure similar to LeNet ([**?**]): two convolutional layers (filter size 32), maxpool layer, two convolutional layers (filter size 64), dropout layer (dropout rate .5), flatten layer, dense layer (128 neurons), dropout layer (dropout rate .5), dense layer with softmax activation. The output of the dense layer is reshaped (16×8) to use as input for the SPN. We used the SGD optimizer with a learning rate of .001 and as loss function we chose categorical cross entropy. The number of epochs was set to 100, with a batch size of 120.

We set hyperparameters by assessing accuracy on the validation set for several values of the hyperparameters. For the learning rate we tried .001 to .1 with step size of .001, each with a dropout rate of .25 and a dropout rate of .5. Aftert fixing the learning rate and the dropout rate, we tested three different numbers of neurons for the dense layer (128, 256, and 512) but since this did not seem to affect the accuracy, we chose the smallest of the tested sizes.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

For Bayesian Batch, we directly used the authors' code **?**, since their architecture is totally different from the ones used by the other tested methods.

1.3 CNN for CIFAR-10

For all methods except Bayesian Batch, the CNN structure used for our CIFAR-10 experiments is VGG [**?**], and consists of a convolutional layer (filter size 64), batchNormalization layer, convolutional layer (filter size 64), batchNormalization layer, maxpool layer, dropout layer (dropout rate .2), convolutional layer (filter size 128), batchNormalization layer, convolutional layer (filter size 128), batchNormalization layer, maxpool layer, dropout layer (dropout rate .3), convolutional layer (filter size 256), batchNormalization layer, convolutional layer (filter size 256), batchNormalization layer, convolutional layer (filter size 256), batchNormalization layer, maxpool layer, dropout layer (dropout rate .4), convolutional layer (filter size 512), batchNormalization layer, convolutional layer (filter size 512), batchNormalization layer, convolutional layer (filter size 512), batchNormalization layer, maxpool layer, dropout layer (dropout rate .5), flatten layer, dense layer (with 512 neurons), batchNormalization layer, dropout layer (dropout rate .5), softmax layer with 10 neurons.

The output of the dense layer was reshaped (16×32) as input for the SPN. We used SGD with learning rate .001 as optimizer and categorical cross entropy as loss function. The number of epochs was set to 250, with a batch size of 100.

We took the VGG model directly from [**?**]. Since the structure was established, we did not attempt hyperparameter tuning as much as with the model used for MNIST. We only tried the learning rates .01 and .001, as well as the dropout rates .25, .3 and .5 for the two last dropout layers. For the number of neurons in the dense layer we tried 256 and 512.

For Bayesian Batch, we directly used the authors' code **?**, since their architecture is totally different from the ones used by the other tested methods.

1.4 CNN for Fashion-MNIST

For all methods apart from Bayesian Batch, the structure and hyperparameter settings were the same as for the CIFAR-10 experiments, except that the learning rate was .0005.

We did not attempt to tune the dropout rate at all (keeping it the same as for the CIFAR-10 data), and tried only three different values for the learning rate (.1, .001, and .0005), picking .0005 as it performed best on the validation set.

For Bayesian Batch, we directly used the authors' code **?**, since their architecture is totally different from the ones used by the other tested methods.

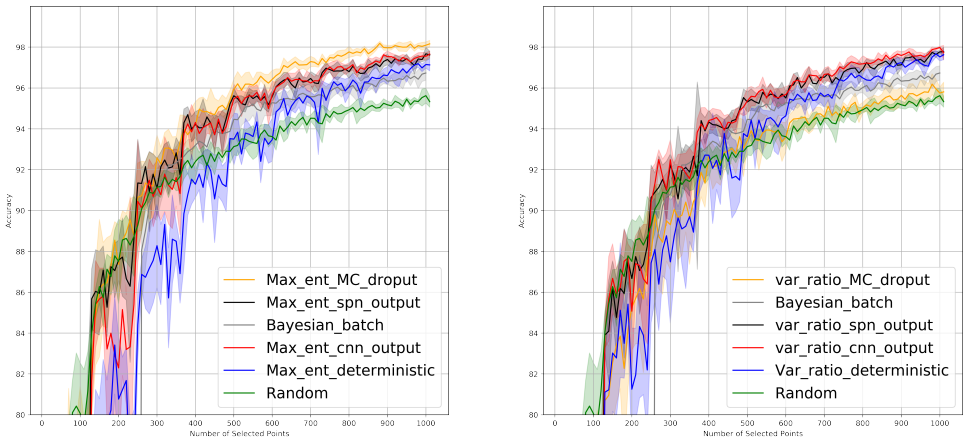
2 Result plots with standard deviation

This section presents the same plots as in the main paper, but with standard deviation indicated in addition. For the sake of readability, we did not show standard deviation in the plots in the main paper. As one can see here, Figures 1, 2, and 4 are not as easy to analyze as the corresponding figures in the main paper. Therefore, for each of these three figures, we here added a table showing the numerical values of accuracy and standard deviation plotted in the figures. For MNIST, Figure 1 plots accuracy at 100 different x-values; here we chose 10 equidistant values for the corresponding table (Table 1). Note that, in accordance with the results reported in the main paper, the standard deviation was calculated over five runs for MNIST and CIFAR-10, and over three runs for Fashion-MNIST.

3 Code

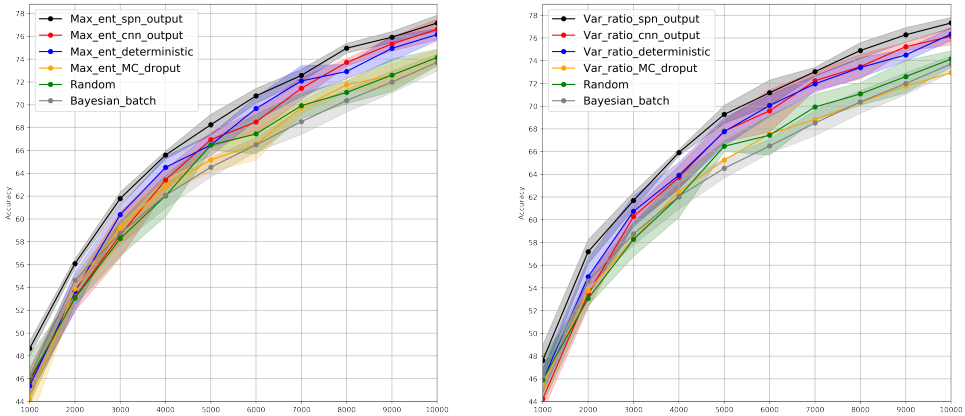
The code from our experiments is included in a zip file. Code that we used off the web is the following

- For MC Dropout we used the code from [**?**]. We modified the CNN in their code to be the same as the CNN that we use in SPN-CNN, *Random* and *Deterministic*.
- For Bayesian Batch, we used the code from [**?**].



(a) Max Entropy (b) Variational Ratio

Figure 1: Accuracy results and error bars on the MNIST data, averaged over five runs, with std dev indicated.



(a) Max Entropy (b) Variational Ratio

Figure 2: Accuracy results and error bars on the CIFAR-10 data, averaged over five runs, with std dev indicated.

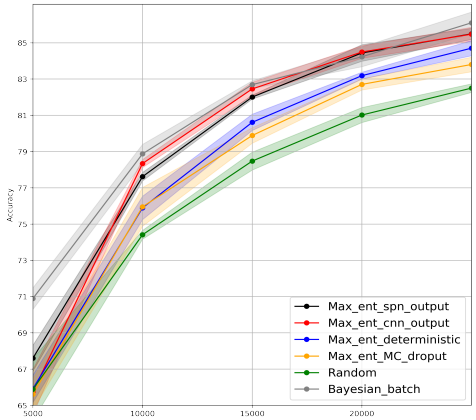


Figure 3: As in Figure 2, but with initial training set size and acquisition batch size of 5000.

Method	100	200	300	400	500
SPN-CNN (SPN, max-ent)	74.29 ± 1.65	86.67 ± .98	92.23 ± 1.46	94.34 ± 0.63	95.31 ± 0.39
SPN-CNN (CNN, max-ent)	62.62 ± 4.68	82.30 ± 4.90	91.16 ± 1.29	94.31 ± 0.37	95.46 ± 0.74
Deterministic (max-ent)	60.25± 5.09	80.77± 3.08	88.28± 2.06	91.29± 0.64	93.49± 1.28
MC Dropout (max-ent)	76.62± 4.77	87.37± 1.93	92.58± 1.12	94.2± 1.12	95.94± 0.50
SPN-CNN (SPN, var-ra)	74.29± 3.64	86.67± 1.89	92.23± 0.53	94.34± 0.57	95.31± 0.49
SPN-CNN (CNN, var-ra)	58.98± 9.66	85.07± 3.01	92.49± 0.98	94.41± 0.42	95.37± 0.63
Deterministic (var-ra)	63.64± 6.80	81.25± 2.99	87.49± 3.00	92.7± 0.96	93.62± 0.44
MC Dropout (var-ra)	74.4±2.97	85.26±2.07	89.31±2.22	92.68± 0.71	93.43± 0.36
Bayesian Batch	9.86±0.69	10.20±0.73	90.79±1.27	93.19±0.76	94.46±0.65
Random	80.06±2.47	87.5± 2.06	91.14±0.39	92.4±0.83	93.13± 0.41
Method	600	700	800	900	1000
SPN-CNN (SPN,max-ent)	95.51 ± 0.27	96.524 ± 0.34	96.92 ± 0.34	97.13 ± 0.26	97.77 ± 0.32
SPN-CNN (CNN, max-ent)	95.67± 0.64	96.24 ± 0.45	97.00 ± 0.22	97.53 ± 0.28	97.55 ± 0.18
Deterministic (max-ent)	93.47± 1.05	95.50± 0.29	96.07± 0.45	96.61± 0.43	97.15 ± 0.18
MC Dropout (max-ent)	96.42± 0.44	97.30± 0.25	97.77± 0.27	97.97± 0.13	98.09± 0.16
SPN-CNN (SPN, var-ra)	95.51± 0.31	96.52± 0.34	96.92± 0.24	97.13± 0.18	97.77± 0.06
SPN-CNN (CNN, var-ra)	95.68± 0.40	96.67± 0.31	97.18± 0.39	97.59± 0.19	97.98± 0.05
Deterministic (var-ra)	94.63± 0.98	95.57±0.53	96.48±0.15	97.24±0.27	97.52±0.2
MC Dropout (var-ra)	93.59±0.71	94.57±0.24	95.03±0.27	95.64± 0.40	95.75±0.29
Bayesian Batch	95.28±0.61	95.57±0.49	96.03±0.26	96.33±0.16	96.73±0.35
Random	93.62± 0.28	94.52±0.15	94.93±0.13	95.17± 0.18	95.62±0.36

Table 1: Accuracy and standard deviation for MNIST over five runs, for 10 selected (equidistant) acquisition steps. This table corresponds to Figure 1.

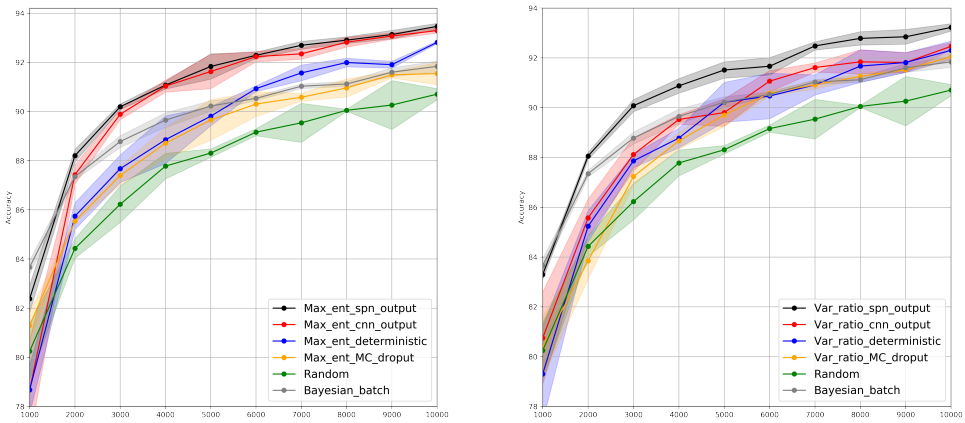
Method	1000	2000	3000	4000	5000
SPN-CNN (SPN, max-ent)	48.65 ± 0.66	56.09 ± 0.42	61.79 ± 0.56	65.59 ± 0.35	68.25 ± 0.92
SPN-CNN (CNN, max-ent)	45.77 ± 1.16	53.13 ± 1.18	58.61 ± 1.99	63.41 ± 0.95	66.95 ± 0.63
Deterministic (max-ent)	45.39± 0.83	53.65± 1.71	60.37± 0.74	64.52± 0.97	66.48± 0.94
MC Dropout (max-ent)	44.19± 2.03	53.87± .9	59.28± 1.41	62.85± 0.65	65.17± 1.04
SPN-CNN (SPN, var-ra)	47.62± 1.36	57.18± 1.06	61.69± 0.73	65.92± 0.27	69.27± 0.79
SPN-CNN (CNN, var-ra)	44.23± 1.07	53.48± 1.23	60.28± 0.68	63.73±1.03	67.8± 0.96
Deterministic (var-ra)	45.84± 1.01	54.98± 1.61	60.74± 1.35	63.92± 1.13	67.76± 1.23
MC Dropout (var-ra)	45.45±1.31	53.75±0.84	58.33±0.57	62.49± 0.55	65.26± 1.6
Bayesian Batch	46.25±0.65	54.63±0.5	58.76±0.98	62.07±1.11	64.53±0.84
Random	45.92±2.1	53.07± 0.6	58.28±1.5	62.02±1.8	66.47± 0.45
Method	6000	7000	8000	9000	10000
SPN-CNN (SPN,max-ent)	70.79 ± 0.65	72.58 ± 0.54	74.96 ± 0.42	75.92 ± 0.49	77.17 ± 0.69
SPN-CNN (CNN, max-ent)	68.49± 1.02	71.44 ± 0.97	73.72 ± 0.31	75.33 ± 0.62	76.61 ± 0.9
Deterministic (max-ent)	69.67± 1.02	72.09± 1.35	72.92± 0.62	74.95± 1.1	76.16 ± 0.52
MC Dropout (max-ent)	66.54± 1.38	69.75± 0.54	71.74± 0.62	72.59± 0.99	74.13± .7
SPN-CNN (SPN, var-ra)	71.18± 1.12	73.04± 0.38	74.9± 0.7	76.28± 0.64	77.32± 0.48
SPN-CNN (CNN, var-ra)	69.59± 1.85	72.24± 0.81	73.48± 1.09	75.24± 0.36	76.15± 0.77
Deterministic (var-ra)	70.06± 0.96	71.96±0.72	73.4±0.87	74.5±0.53	76.34±0.55
MC Dropout (var-ra)	67.55±1.03	68.83±0.7	70.33±0.84	71.76± 0.45	72.94±1.02
Bayesian Batch	66.5±0.62	68.5±1.11	70.38±1.00	71.99±0.96	73.71±0.73
Random	67.45± 1.74	69.92±1.13	71.08±.99	72.6± 1.42	74.15±0.72

Table 2: Accuracy and standard deviation for CIFAR-10 over five runs, after processing 1000, 2000, . . . , 10,000 data points (i.e., for the initial training data set and the subsequent 9 batch acquisitions). This table corresponds to Figure 2.

- In our code we used part of the code in [**?**] for preparing the initial set and new training set in each acquisition.

Method	1000	2000	3000	4000	5000
SPN-CNN (SPN, max-ent)	82.38 ± 0.56	88.2 ± 0.26	90.2 ± 0.11	91.07 ± 0.16	91.83 ± 0.51
SPN-CNN (CNN, max-ent)	78.67 ± 2.05	87.43 ± 0.39	89.89 ± 0.22	91.03 ± 0.25	91.62 ± 0.69
Deterministic (max-ent)	78.68± 0.98	85.74± 0.54	87.67± 0.56	88.85± 0.97	89.81± 0.39
MC Dropout (max-ent)	81.29± 0.56	85.54± .04	87.39± 0.39	88.71± 0.63	89.66± 0.83
SPN-CNN (SPN, var-ra)	83.3± 0.2	88.06± 0.14	90.08± 0.25	90.87± 0.29	91.51± 0.33
SPN-CNN (CNN, var-ra)	80.75± 1.85	85.58± 0.75	88.12± 0.68	89.53± 0.19	89.80± 0.55
Deterministic (var-ra)	79.03± 1.92	85.24± 0.63	87.86± 0.27	88.78± 0.39	90.22± 0.8
MC Dropout (var-ra)	80.40±1.23	83.85±0.73	87.24±0.32	88.67± 0.33	89.71± 0.46
Bayesian Batch	83.67±0.28	87.35±0.13	88.78±0.24	89.65±0.30	90.23±0.21
Random	80.25±1.14	84.43± 0.40	86.23±0.74	87.87±0.52	88.31± 0.17
Method	6000	7000	8000	9000	10000
SPN-CNN (SPN,max-ent)	92.27 ± 0.13	92.7 ± 0.16	92.9 ± 0.13	93.13 ± 0.17	93.46 ± 0.12
SPN-CNN (CNN, max-ent)	92.23± 0.2	92.35 ± 0.22	92.81 ± 0.2	93.07 ± 0.12	93.29 ± 0.1
Deterministic (max-ent)	90.93± 0.11	91.57± 0.32	92± 0.18	91.9± 0.14	92.81 ± 0.05
MC Dropout (max-ent)	90.3± 0.51	90.57± 0.16	90.96± 0.36	91.5± 0.12	91.54± 0.45
SPN-CNN (SPN, var-ra)	91.66± 0.35	92.48± 0.17	92.78± 0.27	92.84± 0.30	93.22± 0.15
SPN-CNN (CNN, var-ra)	91.06± 0.42	91.61± 0.18	91.84± 0.49	91.82± 0.40	92.46± 0.15
Deterministic (var-ra)	90.47± 0.91	90.91±0.40	91.67±0.66	91.81±0.40	92.3±0.37
MC Dropout (var-ra)	90.58±0.1	90.92±0.18	91.26±0.11	91.53± 0.03	92.04±0.18
Bayesian Batch	90.54±0.1	91.03±0.1	91.12±0.10	91.60±0.17	91.84±0.23
Random	89.16± 0.14	89.54±0.80	90.05±0.04	90.26± 0.99	90.71±0.23

Table 3: Accuracy and standard deviation for Fashion-MNIST over three runs, after processing 1000, 2000, . . . , 10,000 data points (i.e., for the initial training data set and the subsequent 9 batch acquisitions). This table corresponds to Figure 4.



(a) Max Entropy (b) Variational Ratio

Figure 4: Accuracy results and error bars on the Fashion-MNIST data, averaged over five runs, with std dev indicated.