Sampling for Deep Learning Model Diagnosis (Technical Report)

***Abstract*—Deep learning (DL) models have achieved paradigm-changing performance in many fields with high dimensional data, such as images, audio, and text. However, the black-box nature of deep neural networks is a barrier not just to adoption in applications such as medical diagnosis, where interpretability is essential, but also impedes diagnosis of under performing models. The task of diagnosing or explaining DL models requires the computation of additional artifacts, such as activation values and gradients. These artifacts are large in volume, and their computation, storage, and querying raise significant data management challenges.**

Here, we articulate DL diagnosis as a data management problem, and we propose a general, yet representative, set of queries to evaluate systems that strive to support this new workload. We further develop a novel data sampling technique that produce approximate but accurate results for these model debugging queries. Our sampling technique utilizes the lower dimension representation learned by the DL model and focuses on model decision boundaries for the data in this lower dimensional space. We evaluate our techniques on one standard computer vision and one scientific data set and demonstrate that our sampling technique outperforms a variety of state-of-the-art alternatives in terms of query accuracy.

I. Introduction

Deep learning (DL) models have enabled unprecedented breakthroughs in developing artificial intelligence systems for analyzing dense, high-dimensional data, such as text, audio, and images. Deep learning is a subset of machine learning methods that use multiple layers to progressively extract higher level features from raw input. Deep learning models have become an indispensable tool for a wide range of tasks, such as image classification, object recognition, speech analysis, machine translation, and more. The task of diagnosis for these purportedly black-box models requires additional artifacts, such as activations. These additional artifacts must be generated, stored, and queried for each DL model being debugged. The addition of these artifacts, which can be up to three orders of magnitude larger than the input data size for each model being diagnosed, turns the process of building, diagnosing, and selecting DL models into a large-scale data management challenge. Diagnosing DL models requires access to artifacts, such as model activations and gradients. Activation values, or *activations*, are learned representations of input data. *Gradients* are partial derivatives of the target output (e.g., the true label of the input data) with respect to the input data. At a high level, activations and gradients are high dimensional vectors with sizes that depend on input data dimensionality and DL model architecture. While activations depict what the deep learning model *sees*, gradients depict *areas of high model sensitivity*.

As previously noted, the naive solution of model diagnosis is pre-computing and storing all artifacts required for model diagnosis scales as the product of the size of input data and number of parameters of the deep learning model. Although the total number of artifacts for small datasets and models is manageable, an overhead that is three orders of magnitude larger than the input data per model is not scalable. On the other hand on the fly generation of these artifacts requires tens of second to several minutes. This makes it difficult to efficiently perform diagnosis tasks, often preventing interactive diagnosis. Thus, with hundreds of gigabytes of artifacts per model, building, diagnosing, and selecting a DL model becomes a large-scale data management challenge.

Sampling is a fast and flexible database technique for approximate query processing, it works well in high dimensions [1], [3], [6], [18] and is a potential candidate for this workload. Top-k maximally activated neurons and the distribution of maximally activated neurons form a large subset of the DL model diagnosis queries (see Section III in addition to the average values of neurons. e.g. *What are the top-10 maximally activated neurons for layer* $conv2$ *for all incorrectly classified objects for* $model_A$*?* The sample in this case would be a subset of data items from the test and training data sets. The top-k queries are an important area of database research. The best known general-purpose algorithm for identifying top-k items is the Threshold Algorithm [13], which operates on sorted multi-dimensional data required to compute the top-k elements. Approximate algorithms for top-k retrieval require building probabilistic models to fit the score distribution of the underlying data as proposed in [50]. However, we wish to avoid computing and storing activations for the entire dataset, which is three orders of magnitude larger than the input data set for each model being diagnosed.The key challenge therefore is in computing a good sample. As we show in this chapter uniform random sample and even a stratified sample do not perform well. Additionally, we seek to avoid computing and storing and analyzing the distribution of activation values for the entire dataset to create a sample. Instead, we leverage the lower dimensional representation of data learned to select a sample for DL model diagnosis (see Section IV for details). Our approach not only reduces the storage footprint and speeds-up queries since we store less data, but it also speeds-up the overall diagnosis process by saving the time it would otherwise take to generate all artifacts for the entire dataset. Specifically, our contributions include:

- Characterizing requirements of DL model diagnosis by studying debugging queries in the literature. We further

develop a simple benchmark for this novel workload by generalizing individual queries used in model debugging papers into query sets that cover families of queries (Section III).
- Presenting a new technique for creating samples for DL model diagnosis (Section IV).
- Evaluating our approach on two datasets and demonstrating its performance compared with a variety of state-of-the-art alternatives.

Our sampling technique can be used to debug any deep learning model where a lower dimensional representation of the input data is learned in a supervised, semi-supervised or unsupervised manner.

II. Preliminaries

We now summarize current approaches for DL model diagnosis and their associated data management challenges, which we address in this paper.

A DL model takes as input a vector $x = [x_1, \ldots, x_N], \in R^N$ and produces as output another vector $S(x) = [S_1(x), \ldots, S_C(x)]$, where C is the total number of output neurons. DL models are constructed in layers, intermediate layers are called *hidden layers*, and each hidden layer of the model is vector-valued. The dimensionality of these hidden layers determines the width of the model, and the number of hidden layers determines its depth. These layers often perform different operations such as convolutions, pooling, dropout, etc. - and are named accordingly. When the model is evaluated over an input data instance, such as an image, it produces a value for each of the C neurons. The raw values thus produced are activations, and derivatives of these values with respect to a target, such as class label, are gradients. Diagnosis of DL models relies on these artifacts. The ML community has a variety of techniques to diagnose these models, which we discuss below.

A. Visualization

Manual visual inspection, is a popular diagnosis technique for DL models [15], [21]–[23], [29]. Standalone tools for visual inspection of DL models built on image data (Cnnvis [29]) and text data (Activis [21], LSTMvis [46]) have been proposed. Some visualization capability is also integrated with deep learning libraries (e.g. Tensorboard [49], etc.). These tools provide static and interactive visualizations of DL model activations and on occasion, gradients. They let ML practitioners view activation or gradient patterns for various layers as well as view aggregates (e.g., average activation) over sets of input data instances belonging to each class, which may be classified correctly or incorrectly. This lets ML practitioners identify specific neuron pattern anomalies and neuron groups and data instances that require further investigation.

Challenge: The size of the artifacts required for these visualizations depends on the size of the input data, and the complexity of the model. It can easily be 3 orders of magnitude larger than the input dataset as shown in Table I. To support interactive visualization for arbitrary queries, these artifacts must be pre-computed since real-time computation is too slow to be interactive. To deal with the associated data explosion, tools such as Activis [21] limit the number of layers that can be visualized in the tool.

B. Examining learned representation

A DL model simultaneously learns a lower level representation of the data and a classifier (in the case of supervised learning). The learned representation (activations of neurons over an input dataset) is used for a variety of goals, such as understanding how a model discriminates between different classes, comparing different model architectures or hyper-parameters, and examining how learning progresses over time by analyzing representations at various checkpoints in the learning process [25], [31], [35].

Challenge: One such analysis [31], takes as input the activations of neurons in two layers from the deep learning model, performs singular value decomposition(SVD) on them, projects activations into the subspace identified by SVD, and computes canonical correlation to find directions in these subspaces that are most aligned. These category of analyses require activations for the *entire* model(s) over the *entire* input dataset. If the training process is being examined, the activations for multiple checkpoints must be generated and stored. As above, the required artifacts, especially if diagnosing multiple models or multiple checkpoints, can result in a data explosion.

C. Feature visualization and saliency analysis

The feature visualization techniques answer questions about what a DL model or parts thereof are looking for by generating examples from the learned model [36]. *Feature visualization* uses derivatives to iteratively modify an input, such as random noise, with the goal of finding the input that maximally activates a particular neuron(s). *Saliency analysis* identifies parts of the input that have the largest effect on the output. This consists of a number of approaches that propagate gradients through the model to identify areas of highest activation and highest sensitivity [30], [42], [43], [47], [58].

Challenge: Even simple DL models consist of hundreds of thousands of neurons (e.g. Table I). Finding the appropriate set of neurons to visualize can be beyond the powers of human cognition. Saliency analysis works on a per-input-data-item basis; ML practitioners would need specific input data points, such as images, for these methods. DL datasets consist of tens of thousands of instances, picking appropriate data instances from these large datasets is imprecise, especially if the dataset is new, large and contains unexplored scientific data.

D. Statistical analysis

Many datasets are annotated. Language models are annotated with parts of speech or linguistic features and image datasets are annotated with object information. For instance, Broden dataset [7], has pixel-level annotations that indicate the object to which each pixel belongs. These annotations are used to pose hypotheses and conduct statistical analyses

dataset name	Image size (KB)	No. model params	Ratio artifacts:input data
MNIST	0.78	107,786	53 ×
Galaxy Zoo2	1.3	1,095,842	2905.5 ×

TABLE I: Data size and model sizes for standard ML dataset (MNIST) and scientific image dataset (Galaxy Zoo2).

between neuron(s) activations and annotation to evaluate these hypotheses [41]. We include statistical analysis as it is important technique for model diagnosis and interpretability.

Challenge: Statistical analyses require such annotations to formulate hypotheses. The two datasets we utilize do not have any annotations. Indeed, most scientific image datasets do not, which makes statistical analysis impossible,

III. Workload Characterization

We now develop a summary workload that captures the requirements of a large set of DL model diagnosis queries. Model diagnosis techniques, such as visualization and examination of learned representation, bring the number of neurons and data instances to be examined to a smaller and manageable number. This section focuses on queries from these two categories, as these queries helps make downstream analysis, such as feature visualization, attribution and saliency analysis tractable over massive datasets. We do not include queries from statistical analysis as it requires annotations on the dataset.

ML practitioners typically start model diagnosis with techniques utilized by visualization tools from Section II. A common practice is to create data subsets that are incorrectly classified, generating aggregates (such as average activations, top-k highest activations etc.), and compare them to similar aggregates for data instances that were correctly classified for each class. ML practitioners start the analysis from sets [21], [29], such as all incorrectly labeled instances of $class_a$, rather than specific instances to find such patterns. This analysis helps them identify important patterns for the various subsets and reduce to a manageable number both data instances and parts of the model (layers and neurons) to be examined [21], [29]. Based on this analysis ML practitioner can now start correlating input data and parts of the model, conducting attribution and saliency analyses. Similarly, the common practice when comparing two different models trained on the same data leverage techniques listed in examining learned representation from Section II. For instance, ML practitioners generate neuron activation values for each data item for both models for each layer and then compare these to the logits for each class learned by the respective model to decipher each model's rate of learning to understand the impact of additional layers and their sizes and thus diagnose how complex the model must be to complete this task.

Table II lists representative queries from the literature used to diagnose DL models. We make two observations from this list of queries. First, DL model diagnosis queries require one of three quantities: the top-k maximally activated neurons, the distribution of maximally activated neurons or the average activation values. The focus on maximal and top-k values as opposed to minimal values is due to activation functions [2] used in DL models. Without such functions DL models are just complicated linear regression models. ReLU is the most commonly used activation function today [39]. It removes negative values and propagates positive values. Mathematically, ReLU is defined as $max(0, val)$. Therefore, in the DL literature sample queries often focus on average or maximal values but not minimum values. Thus, to characterize an ML diagnosis workload instead of focusing on all aggregates we focus on three aggregates (1)Top-k maximally activated neurons, (2) Average activation values for neurons and (3) distribution of maximally activated neurons.

Second, each query in Table II is part of a family of queries. For instance, the answer to Q1 requires average values of all neurons for a specific layer ($conv2$) for all classes. A family of queries for Q1 would include average values of neurons for *any layer* and *any class* where data instances could be *correctly* or *incorrectly* classified. We can see that queries Q3 and Q1 belong to the same family. Similarly, Q2 belongs to a family of queries that require top-N neurons, across classes, layers, incorrect, and correct classification. Thus, to characterize this workload we utilize the entire family of queries. We call these families of queries *query sets*.

We now introduce some notation and define query sets formally.

A DL model M is a vector of N units or neurons. M is learned and tested over data D. Artifacts, such as activations A, are vectors of the same dimensionality as M, computed over data D. a_{id} is the activation value(s) of a set of i neurons, where $i \subseteq N$, on d^{th} data item(s), where $d \subseteq D$. A query computes a measure ϕ, such as the mean, top-K, count, or count of maximum values for a_{id}. A query set S is a set of queries that enumerate over all vectors of activation values a_{id} that correspond to all layers, all classes and both correct and incorrect classifications. Given the preceding notation we can define a DL model diagnosis query set:

Definition 1. *A query set $S(a_{id}, \phi)$ is a set of queries, where $i \subseteq N, d \subseteq D$, and ϕ is a measure.*

Instead of evaluating our techniques on specific queries from Table II, we utilize the three query sets shown in Table III to characterize DL model diagnosis workload. These query sets include all queries of a specific family. We leverage these query sets to measure effectiveness of sampling techniques to ensure these techniques do well on the entire family of queries represented by the query set, not just on individual queries.

Query Q2 and all queries of this family are represented by query set S1, which computes the top-K maximally activated neurons. Queries Q1, Q3, and others of this family are repre-

QN.	Queries
Q1.	What is the average value for all neurons for layer $Conv2$ in $model_A$ across all classes? [15], [21], [31]
Q2.	What are the top k maximally activated neurons for layer $Conv2$ for all incorrectly classified objects for $model_A$? [21], [23], [29]
Q3.	What is the average neuron activation pattern for the last hidden layer in $model_A$ for incorrectly classified $class_a$ vs. correctly classified $class_a$? [21], [23], [44]
Q4.	Compute the similarity between the logits of $class_a$ and the representation learned by the last convolution layer by $model_A$? [26], [31]
Q5.	For images of $class_a$ classified as $class_b$, what are all of the maximally activated neurons in the last convolutional layer? [21], [29]
Q6.	Does $model_C$ learn a representation for $class_e$ faster than it learns the representation for $class_f$? [25], [31]
Q7.	How similar are the representations learnt by two different model architectures, $model_A$ and $model_B$, on the same dataset? [26], [31], [35]

TABLE II: Representative queries for deep learning diagnosis workload.

Query Sets
S1. Set of top-K maximally activated neurons.
S2. Average activation values of neurons.
S3. Distribution of maximally activated neurons.

TABLE III: Query sets for deep learning model diagnosis workload.

sented by query set S2, which computes the average activation for neurons. Queries Q4, Q5, Q6 and Q7, and others of their family are jointly represented by query sets S2 and S3, because finding similarity depends on the average neuron values and the maximally activated neuron distribution.

Query sets can consist of any combination of neurons and data items. Instead of considering this immense set of combinations, we limit our evaluation to all combinations of layer, class and classification (correct or incorrect). Thus, to measure accuracy of a query set for a sample, we first compute the query results for each of these combinations (layer, class and classification). Next, we compute a metric comparing the results from the sample with the results for the same combination on the entire dataset. Our comparison utilizes metrics specific to each query set, e.g., precision for S1, cosine distance for S2 and, Jensen-Shannon distance for S3 (we describe these metrics and the rationale for picking them in more detail in Section V). Finally, we calculate the over-all query set accuracy for each query set by averaging the value of the corresponding metric over the combinations.

IV. Approach

To enable interactive model diagnosis, our approach creates a sample. We compute the results of a query set on this sample instead of the entire data. In this section we describe our approach and present other baseline techniques for selecting these samples.

The key insight we utilize to avoid generating and storing activation values is that DL models learn a lower dimension representation of the data, and a classifier. DL training transforms the input data, creating a new representation with each layer. Training criteria encourage training set neighbors, such as data points from the same class, to have similar representations. Leveraging this lower dimensional representation learned by the model has the dual benefit of reducing dimensionality of the data and focusing on the representation learned by the model. Since the objective of the workload is to diagnose this model, we hypothesize that leveraging the learned latent space to select a sample will be key to understanding what the model has learned. For model diagnosis we view the training, and test data points in the **latent space**, i.e., instead of viewing the data in the high dimensional original format of images, audio or text, we utilize this lower dimensional representation of the data learned by the model's last hidden layer to create samples.

Our goal is to diagnose the model, which implies that a subset of the queries will focus on what the model got wrong, as shown in Table II. In a classification problem with multiple classes, the decision boundaries partitions the underlying vector space into multiple regions, one for each class. Decision boundaries are where the output label of a classifier is ambiguous, i.e., where errors and mis-classifications occur. The diagnosis of a DL model requires exploration of the **decision boundary** for a model [16], [55].

For instance, Figure 1 depicts this lower dimensional representation for two datasets we use for evaluation, MNIST [34] and Galaxy Zoo2 [17]. We use a dimensionality reduction technique, t-Distributed Stochastic Neighbor Embedding (t-SNE) [51], to reduce dimensions of this data to visualize it in two dimensions. In Figure 1, each point represents an image from the test set, and colors indicate the true class labels. We make two observations from this visualization: (1) the data representations in latent space show separation for each class, and (2) most mis-classified instances are on the edges of data points groupings.

Thus, it is at the decision boundary, we find most of the incorrectly classified data points examples and the diverse correctly classified data items. Farthest from the boundary we find the correctly classified data instances, where their learned lower dimensional representation are very similar.

Our approach is based on utilizing the lower dimensional representation when selecting data items for our sample, and focusing on decision boundaries in the *latent space* when selecting the data points to include in our sample.

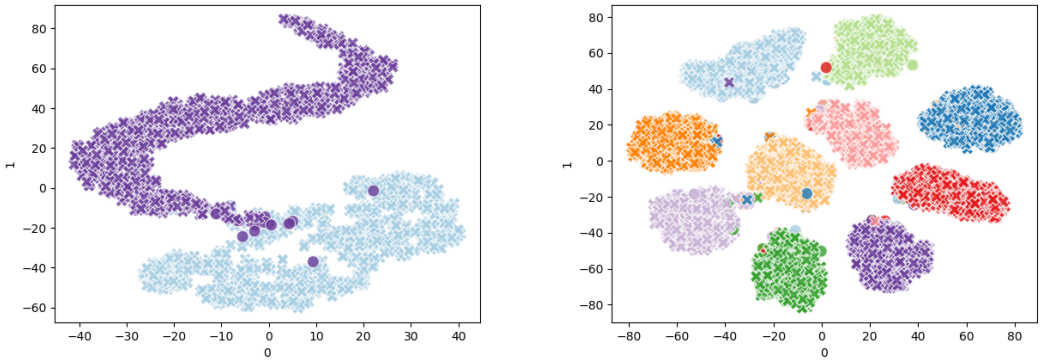


Fig. 1: T-SNE representation of test data or Galaxy Zoo2 (left) and MNIST (right) from the last hidden layer. Each data point represents an input image from the test set. Data point colors represent the true labels.

A. Baselines

In the AQP literature the two most popular sampling techniques are uniform sampling and stratified sampling. We consider both of these sampling techniques as baselines. Uniform sampling can directly be applied to our problem without changes. Stratified sampling partitions data into groups called strata and computes a separate sample for each stratum.

In database systems such as BlinkDB [3], strata are defined over a subset of columns that typically correspond to categorical valued attributes, e.g. `city`. For DL model diagnosis, the underlying data can be considered a relation, with each row representing a data item (e.g., one image) and each column a value of interest, such as the activation value for a neuron in the model. Each row can be extended with metadata, such as the predicted class and the class label. To implement stratified sampling for our use case we propose *stratified by CM* as a baseline technique. Because of the large dimensionality of the data, we cannot create a stratum for each possible value in each column, instead we use the classification result or the metadata associate with each data item to create the strata. Each data point is classified by the model as belonging to a class. This result is encapsulated in a *confusion matrix* (a.k.a. the error matrix), which tabulates the performance of a classification algorithm. For a binary classifier, the confusion matrix counts the number of true positives, false positives, true negatives, and false negatives. For multiple labels, the confusion matrix generalizes this concept. Each row of the matrix represents a predicted class, while each column represents a true class. In this technique, we sample based on cells in the confusion matrix. We call this technique *stratified by confusion matrix (CM)*.

Additionally, since we utilize the latent space for creating the sample we also consider a baseline which selects a sample based on naively sampling from the latent space. This baseline covers the n-dimensional latent space by creating grid in this space and then sample from each each partition. Our goal for this baseline is to ensure our sample contains instances of data that lay in different regions of that latent space. However, the latent space we choose is high dimensional, e.g., for the MNIST dataset, the latent space is $84D$. Even if we divide each dimension into two buckets, we get a total of $2^{84} \sim 1.93e+25$ buckets. Instead, we reduce the dimensionality of data in the latent space for this analysis using PCA. We call this naive technique *simple latent space* sampling. For this sampling technique, we collect equal number of instances at random from each underlying grid.

In addition, we use two other techniques from the literature as baselines. First, we use visualization aware sampling (VAS) for large scale data visualization, such as scatter and map-plots. VAS is based on the interchange algorithm [37], which selects tuples that minimize a visualization-inspired loss function. Visualization-inspired loss is based on three common visualization goals: regression, density estimation and clustering. The interchange algorithm creates a sample that maximizes visual fidelity of the data at arbitrary zoom levels.

Second, we use explicable boundary (EB) trees [54] to create a single sample from input data. This method constructs a boundary tree to approximate the complicated deep neural network models with high fidelity. EB trees provide a single sample for a dataset and a model which explains the boundary between each class learned by the DL model.

We describe details of sample selection from baselines in Section V-C.

B. Clustering in Latent Space

An important part of our approach to selecting a sample for DL model diagnosis is to ensure that model decision boundaries are represented in the sample. As described earlier, model decision boundaries are where the model makes mistakes and when diagnosing the model, we want to focus on this area of the latent space. To determine boundaries in latent space, we cluster data in latent space and fit a model to estimate the parameters for each class in that space. We do this in both supervised and unsupervised manner. When fitting a supervised model, we use the class labels. In the unsupervised

Algorithm 1: Algorithm for selecting sample.

Data: input data in latent space, f,k, j
// k num class labels, f is sample size
$Clusters \leftarrow None$
$sample \leftarrow None$
Clusters = ClusterAndSortData(data,k)
foreach $cluster_i$ *in Clusters* **do**
  $s1 \leftarrow data.head(f * j)$
  $s2 \leftarrow data.tail(f * (1 - j))$
  $sample \leftarrow s1 + s2 + sample$
end
return sample

case, we use parameterized models so we utilize the number of unique classes present.

In both supervised and unsupervised cases the models fitted to the latent space provide us with the likelihood that and object belongs to a class or cluster. For binary classification to determine whether an object belongs to class A or class B, let $P(A|x_i)$ be the likelihood that a data instance x_i belongs to class A. In this case, the points on the decision boundary of class A and class B are those for which the ratio $\frac{P(A|x_i)}{P(B|x_i)}$ is ≈ 1. A lower value of likelihood ratio would imply that $P(B|x_i) > P(A|x_i)$ in which case x_i would be assigned to cluster or class B. The higher the likelihood that an object belongs to class A, the higher the ratio $\frac{P(A|xi)}{P(B|x_i)}$ will be.

For a multi-class classifier, where a data point x_i may belong to classes $\subset a, b, c, \ldots$, this ratio would be, $\frac{P(A|x_i)}{\sum_{z \subset b,c,d,\ldots} P(Z|x_i)}$, or

$$\frac{P(A|x_i)}{P(\neg A|x_i)}$$

Our sampling technique clusters the data in the latent space, then sorts data in each cluster or class by the ratio of likelihood of belonging to that particular class. This sorted list thus consists of exemplars on the higher end and outliers on the lower end of the list. We utilize a tuning parameter j to determine the proportion of exemplars and outliers in our sample. We select $j\%$ from the outliers and $1 - j\%$ from the exemplars. Algorithm 1 describes this approach in further detail.

For the unsupervised technique, we utilize a parameterized clustering technique, the Gaussian Mixture Model (GMM). These models offer a probabilistic way to represent normally distributed sub-populations within an overall population. We set the number of clusters in GMM to be equal to the number of unique classes in the dataset.

For the supervised technique, we use max-margin classifiers to classify the data in the latent space. Margin classifiers are a class of supervised classification algorithms that utilize distance from the decision boundary to bound the classifier's generalization of error. Support vector machine (SVM) [48] is an example of this category of classifiers, which learns boundaries based on labels so that the examples of the separate classes are divided by a clear gap that is as wide as possible. SVMs utilize kernel functions [24]; these help to projecting data to a higher dimensional space where points can be linearly separated. DL models do not have non-linear activation functions after the last hidden layer, so the latent representation from last the hidden layer should enable discovery of linear boundaries. Thus, we utilize a linear kernel for SVM [14], which has the dual advantage of being faster than non-linear kernels and less prone to over-fitting. Results of the classifier are turned into a probability distribution over classes by using Platt scaling [38], [56]. These probabilities are used to sort the data items in each cluster or predicted class and then select a sample.

V. Evaluation

In this section we empirically evaluate our sampling techniques which is based on the hypothesis that sampling evenly from the latent space is not sufficient; model decision boundaries are the most important region of this latent space for answering model diagnosis queries; and they must be well represented in a reliable sample.

We evaluated our sampling techniques from Section IV on two different datasets. We first describe metrics we used to evaluate query sets defined in Section III and datasets and DL models we used for experimental evaluation. We then describe the experiments we conducted and results of these experiments in Section V-C, finally we present the analysis of these results in Section V-D.

A. Metrics

In this section we define metrics for evaluating the three query sets defined in Section III. **Query set S1** retrieves the set of top-K maximally activated neurons. To measure how well our sample performs we use *precision* as the metric. Precision is the fraction of top-k results from the sample that belong to the true top-k result. Precision lies between $[0, 1]$. A precision value of 0 implies that the sample top-k does not contain any of the full data top-k neurons.

Query set S2 retrieves the average value of neurons. This is a high dimension vector of floating points, where dimension is the number of neurons in a layer. Additionally, this is a sparse vector, i.e., many neurons may have zero average activation because of non-linear activation like ReLU. We used cosine distance [8] to measure the distance between the average vector for the entire dataset and the average vector from sample due to the properties of high dimensionality and sparseness of the average neuron vectors, which lies between $[0, 1]$. Cosine distance between two vectors A and B is defined as:

$$1 - \frac{A.B}{\|A\|\|B\|}$$

Query set S3 retrieves the distribution of maximally activated neurons. As this is a true distribution an obvious metric would be Kullback-Leibler (KL) divergence [27]. However, we encounter two issues with using this metric. First, KL divergence is unbounded, which means it is not a true metric,

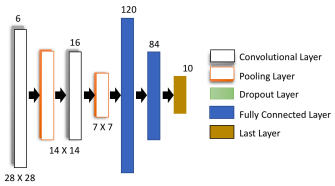


Fig. 2: Deep learning model for MNIST dataset.

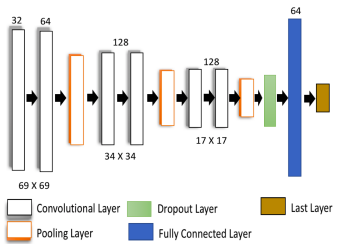


Fig. 3: Deep learning model for galaxy Zoo2 dataset.

and it is difficult to assess how close two distributions were. Second, KL divergence is defined only on distributions with non-zero entries. This is not true for maximally activated neuron distribution, which may have neurons with zero counts. Thus, we used Jensen-Shannon divergence [20] instead, which is based on KL divergence. Jensen-shannon divergence is both symmetric and finite valued. Jensen-Shannon distance is squareroot of Jensen-Shannon divergence which is defined as:

$$\sqrt{\frac{D(p\|m) + D(q\|m)}{2}}$$

where p and q are the two distributions being measured, operator D represents KL divergence, and m is there mixture distribution. It is a true metric and lies in $[0, 1]$.

B. Datasets and Models

We evaluated our sampling techniques on two datasets, Galaxy Zoo2 [17] and MNIST [34]. For each dataset, we built and evaluated one deep learning model. The MNIST dataset consists of 28×28 pixel gray-scale images of handwritten numerical digits with a training and test set of 60K and 10K images, respectively. We trained the six layer neural network depicted in Figure 2. This model is a based on LeNet-5 [28] for classifying MNIST dataset with added batch-normalization after every layer.

Galaxy Zoo2 [17] is a public catalog of $\sim 240,000$ galaxies from the Sloan Digital Sky Survey [40] with classifications from citizen scientists. The Galaxy Zoo decision tree [9] lists the questions answered by citizen scientists. We took a subset of this dataset to classify images that appear edge-on vs face-on, i.e. which of the telescope images show the celestial object facing the telescope or 'face-on' vs. as seen from a side or 'edge-on' (question T01 in [9]). The training and test datasets consist of $54,333$ and 2118 images, respectively, each a 69×69 color image. We trained a model depicted in Figure 3, which is a variation of the model from [11]. In our variation of this model, we reduced the number of dropout layers and added batch normalization after every convolutional layer. We achieved 99% accuracy on the test set and an overall weighted F1 score of 0.99.

We trained the models and extracted activations from them using the TensorFlow [49] library. For both models, we used the representation from the last hidden layer to drive our sampling technique, and the last hidden layer was a fully connected (FC) layer. The MNIST data representation is from layer FC-2 (Figure 2) with 84 neurons. The Galaxy Zoo2 data representation is from layer FC-1 (Figure 3) with 64 neurons.

C. Experiments

For our **first experiment**, we evaluated the three query sets on the two datasets using the metrics described above. For each dataset, we created samples of size 5%, 10%, 20%, 40% and 80% for the eight sampling techniques we are evaluating. Our rationale for choosing sampling techniques is described in Section IV, here, we provide a brief description of each technique:

(1) *Random* sampling draws a sample from the dataset uniformly at random without replacement. (2) *Stratified by CM* sampling contains a sample with data items drawn from each cell of the confusion matrix in proportion to the number of data items in the cell. For instance a 5% sample select uniformly at random, 5% of the data items from each cell in the CM. (3 and 4) *Visually aware sampling (VAS)* and *Explicable Boundary (EB) tree* sampling utilize the techniques presented in [37] and [55] respectively. (5) *Simple latent space* sampling divides the latent space into a grid and then samples equally from each cell in the grid. (6 and 7) *GMM* sampling fits a GMM to the data points in latent space. For each of the resulting clusters, data points belonging to each cluster are sorted by the likelihood ratio $\frac{P(A|x)}{P(\neg A|x)}$ of belonging to that cluster. The sample is then created by selecting data points from the two ends of this list for each cluster, with a tuning factor j determining what fraction is selected from either end. We have two GMM samples since we evaluated impact of two types of co-variance matrices, spherical and full. Finally, (8) *MaxMargin classification* sampling classifies the data points in the latent space with a max margin classifier, sorting points in each class by the ratio of their likelihood belonging to that class, and choosing from the two ends of this list with a tuning factor of j, like the GMM samples.

For the first experiment, we fixed the tuning factor j to 0.7 for GMM and MaxMargin samples. We studied the impact of this tuning factor in the third experiment. The EB tree technique creates a single sample since there is a single boundary tree for a model and corresponding data. Figure 4 shows the results of this experiment for both datasets. As we increased sample size the query set results got increasingly more accurate until, at fraction 1.0 or on the full dataset, the metrics for all sampling techniques were coincident at 1.0 for S1 and 0 for S2 and S3.

For *simple latent space* sampling we reduced the dimensionality of latent space from 84 and 64 to five for both MNIST and Galaxy Zoo2 and then divided each dimension into 2 bins,

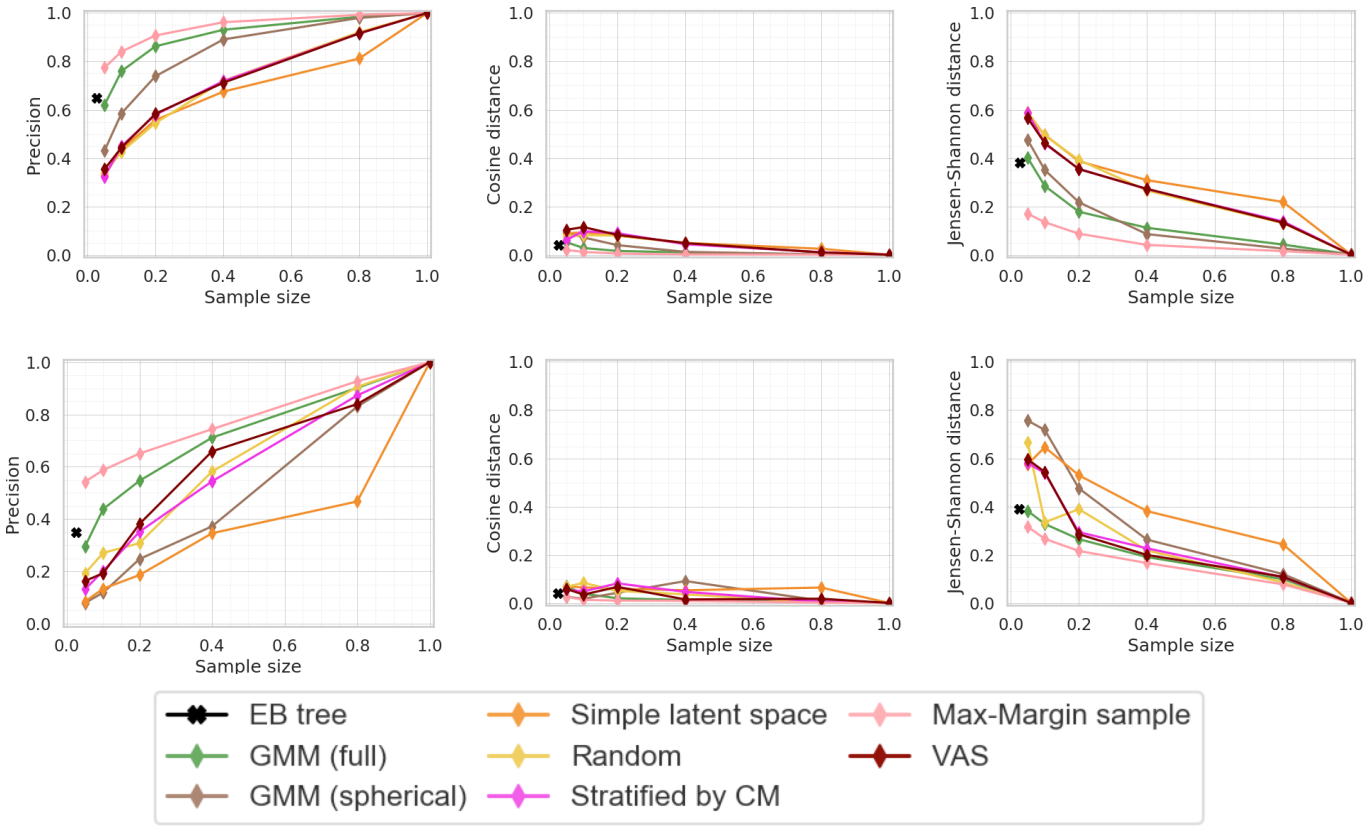


Fig. 4: Performance of query set S1, S2 and, S3 for increasing sample size for various sampling strategies. Top row MNIST, bottom row Galazy Zoo2. From left to right columns, S1, S2, and S3. The X-axis shows the sample size as a fraction of the entire dataset.

resulting in 2^5 or 32 bins. We then sampled equally from each bin. This is the only technique where we sampled equally rather than sample in proportion to the number of items in the bin. We did this in order to evaluate the impact of sampling from the latent space. Interestingly, this technique did not do well on all three query sets. To minimize the impact of randomness, we selected each sample ten times and evaluated it and average results from these ten iterations. As we can see from Figure 4, the *simple latent space* sample behaved as well as the random sample. While this sample provided adequate results on S2, giving on average less than 10% error, its performance on S1 and S3 was not adequate. The knee seen for this sample (at 80% of the dataset) occurred because at this point the sample had the fewest number of data items compared to other samples: data were unevenly distributed in the latent space, and we sampled equally from each bin rather than in proportion to the size of the bin.

The *stratified by CM* sample performed much better than both the *random* and *simple latent space* samples for S1 and S3. *VAS* did as well as *stratified by CM*, this is of note because the VAS sample had no knowledge of classification of each data points and was trying to minimize a visualization-based loss function, which is trying to ensure that that the sample replicated the data density of the original distribution. All three clustering-based samples *GMM (full)*, *GMM (spherical)* and *MaxMargin classification* based samples did better than the baseline samples on all three query sets in most cases. *GMM (full)* did better than *GMM (spherical)* for both datasets. *GMM (full)* fit the data better, as expected, and thus did better on selecting exemplars and outliers when compared to *GMM (spherical)*. The goodness of fit is dependent on the dataset. *GMM (spherical)* does better than *stratified by CM* for the MNIST dataset but worse for the Galaxy Zoo2 dataset. From the two dimensional representation of the data in latent space for the two datasets in Figure 1 we can see a separation between the ten clusters in MNIST, while the two clusters in the Galaxy Zoo2 dataset were not clearly separated. Additionally, for MNIST each cluster appeared to be somewhat symmetrical, but the two clusters for the Galaxy Zoo2 dataset did not have a clear separation, and one of the clusters is highly asymmetrical. *GMM (spherical)* with a isotropic covariance matrix has difficulty fitting the Galaxy Zoo2 dataset. *GMM (full)* fit a more complex gaussian to each cluster, and this, in turn, provided a much better estimation of outliers vs exemplars. This difference can be seen in two datasets. While *GMM (full)* sample did better than *GMM (spherical)* sample for both datasets, the difference in performance was higher for when the underlying data distribution assumptions

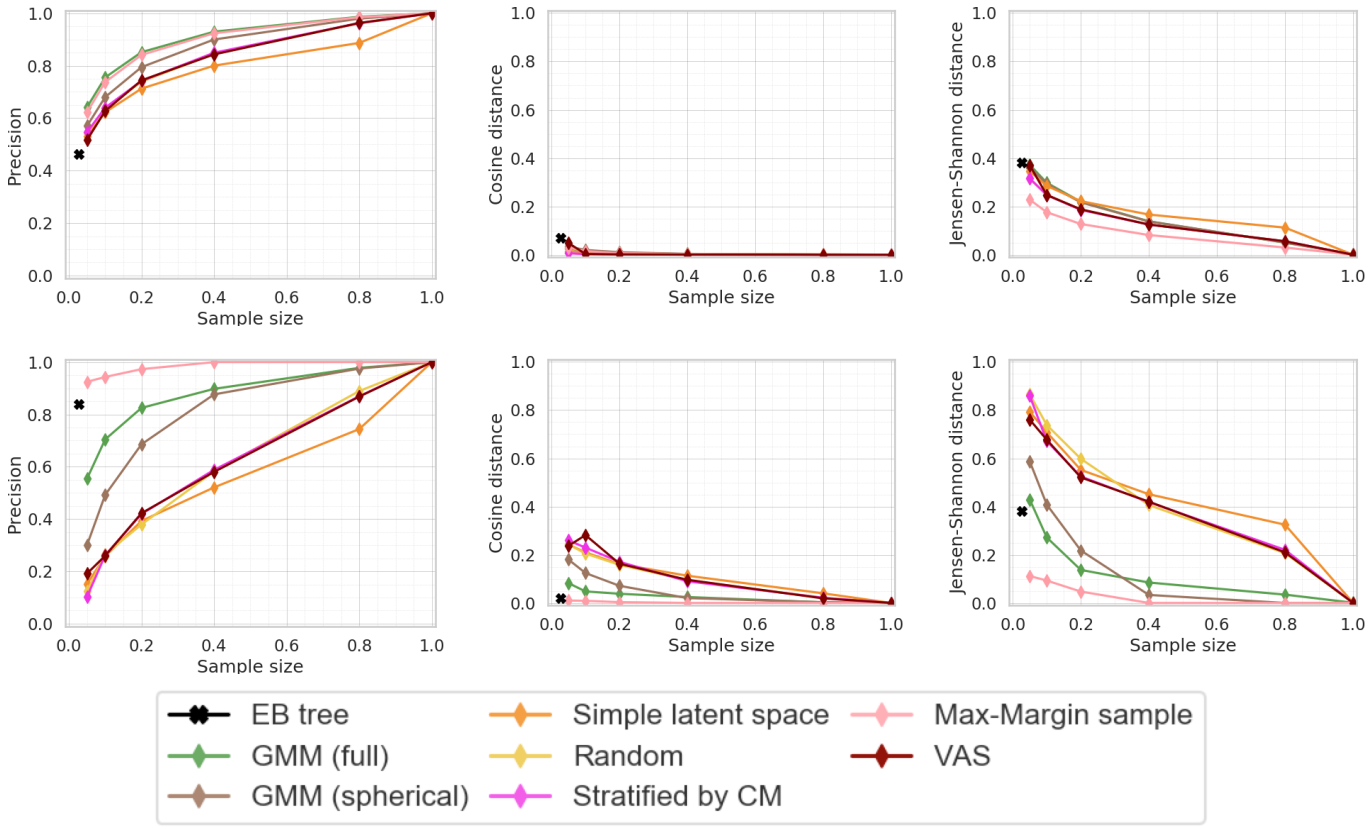


Fig. 5: Performance of query set S1, S2 and, S3 for increasing sample size for various sampling strategies for MNIST dataset. Top row shows the results for correctly classified data items and the bottom row shows results for incorrectly classified items. From left to right columns, S1, S2, and S3. The X-axis shows the sample size as a fraction of the entire dataset.

were not met for *GMM (spherical)*. *MaxMargin classifier* based sampling performed the best on all three query sets. Additionally, Figure 5 shows the results for all from this experiment on the three query sets separated by correctly(top row) and incorrectly(bottom row) classified items, here we can see that while *MaxMargin classifier* performs well on correctly classified data items, it performs exceptionally well on the incorrrectly classified data items. We delve into this further in Section V-D.

Finally, *EB tree* technique provided a single sample since there was only one boundary for model. As expected, it did well picking the outliers and therefore performed well on both S1 and S3. For both datasets, EB-tree based sample was the smallest and performed second best on these two query sets. However, as the EB tree sample focused inordinately on the outliers, it did not perform as well on S2. On the well-separated latent space for the MNIST dataset the *EB tree* performed on par with other sampling techniques. However, for the Galaxy Zoo2 dataset, it did not perform as well. The *MaxMargin classification* based sampling performs better than EB-tree sample for all three queries for both datasets.

We analyze further analyze the results for this experiment in Section V-D and delve into why *MaxMargin classification* based sampling outperforms stratified by CM and other baselines.

In the second experiment, we examined the impact of varying the number of top-k neurons in S1 and measured the precision achieved by each of the eight sampling techniques. We show the results for a 5% sample, with k equal to 5, 10, 20, 50, 100. Results for this experiment are shown in Figure 6.

For the MNIST dataset, a 5% sample had a precision 0.98 for the top-100 neurons. However for the Galaxy Zoo2 dataset this number was much lower, at 0.70 for the top-100 neurons. This is due to two factors: (1) the test dataset, over which we evaluated this query for MNIST was 10k while for the Galaxy Zoo2 dataset it is 2k. A 5% sample was 500 data items for MNIST and 105 items for Galaxy Zoo2 dataset. (2) the model for MNIST had 107,786 parameters or neurons, and Galaxy Zoo2 had an order more parameters at 1,095,842. Thus, a 5% sample for the Galaxy Zoo2 dataset was both smaller and trying to capture a more complex model. This is confirmed by an additional experiment, where we increase the Galaxy Zoo2 sample size to 500 elements, we get 85% coverage on the top-100 neurons.

For both datasets *MaxMargin classification* sampling had the highest precision. *EB tree* was next for both datasets. This is not surprising because EB tree focuses on decision boundaries. This reinforces our hypothesis that decision boundaries

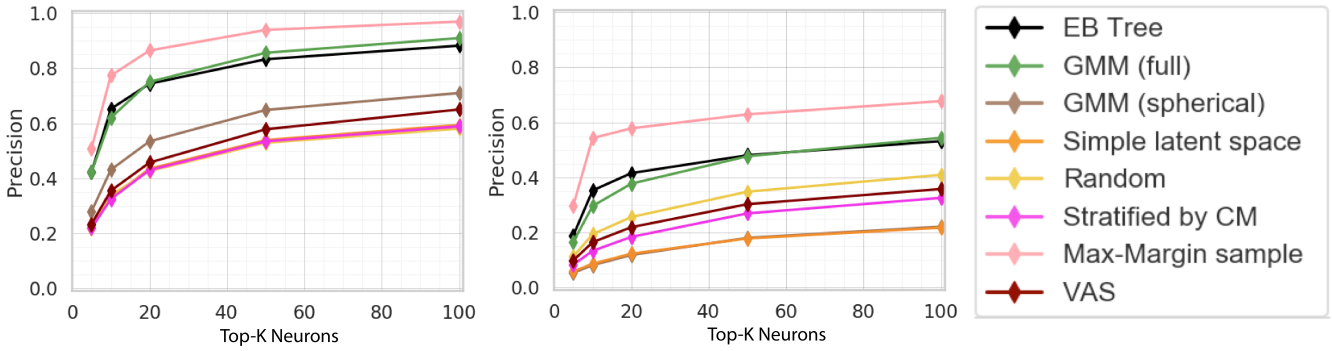


Fig. 6: Precision for query set S1, number of top-k neurons (x-axis) in the 5% sample. Left panel MNIST, right panel Galaxy Zoo2.

need to be well represented for a sample to perform well on model diagnosis queries.

In the third experiment, we evaluated the impact of tuning factor j for three clustering samples *GMM (full), GMM (spherical), MaxMargin*. Tuning factor j is a number between 0 and 1 and is used to determine how many data points in the samples come from the lowest values of likelihood ratio or outliers. We evaluated the impact of this tuning factor on a 5% sample for all three sampling strategies. We evaluate query sets S1, S2 and S3 on tuning factor values of 0, 0.25, 0.50, 0.75 and 1.00. In all three sampling strategies, we picked data items in order from the sorted list for each cluster. Our sample is selected by selecting items from both ends of the sorted list and picking $frac*j$ items from the head or outlier end of the list and, $frac*(1-j)$ from the exemplar end of the list. Thus, for the tuning factor value of 0, all data instances in the sample are picked from the exemplar end of the list and for a tuning factor value of 1 all data instances were picked from the outlier end of the list. In this experiment, we additionally created a weighted sample, where the weight was simply the reciprocal of the likelihood ratio. Likelihood ratio can be unbounded for exemplars, therefore for purposes of numerical stability we selected a threshold. To reduce the impact of random selection, we selected a weighted sample ten times and reported the average value. Figure 7 shows the results of this experiment. For S1, when the dataset contained only exemplars at tuning factor 0, precision was the lowest for the sample. Precision grew as the value of tuning factor increased and plateaued at tuning factor ~ 0.7. The max top-10 precision for the MNIST dataset was 0.8 and galaxy Zoo2 is 0.57. This was the max value for top-10 that can be achieved on a 5% sample with the three sampling techniques for either dataset. For S2, the average activation value was not impacted as much by the tuning factor. The difference was small enough not to significantly impact the value of this metric. For S3, we saw results similar to S1. The highest values were at $j = 0$, because at this point there was the least amount of diversity in the data points; each cluster only contributed exemplar data points. As the number of outliers increased, the distance between the distribution became lower, the lowest point around $j \sim 0.5$, as the tuning factor increased further and the sample contains an increasing number of outliers this value became lower at a slower rate. Weighted sample values are indicated by dashed lines on the Figure 7. The weighted samples did not achieve the best value for any of the queries for either dataset. This indicates that picking the data items based on the likelihood ratio directly provides better results on rather than relying on selecting a sample weighted by the likelihood ratio.

Sampling Technique	MNIST	GZoo2
Entire Test set	9893(107)	2097(21)
Uniform	494(4)	105(1)
Latent space sample	487(9)	108(0)
Stratified by CM	494(11)	105(1)
Stratified Weighted	397(107)	85(21)
GMM (full) sample	464(48)	104(4)
GMM (sph) sample	500(11)	108(0)
Max margin sample	427(85)	89(18)
Visually Aware Sample	492(8)	92(4)
EB Tree sample	198(72)	44(8)

TABLE IV: Number of data points in samples for each sampling strategy with correctly classified (incorrectly classified) data points for 5% of the sample from the test set in both data sets.

D. Performance Analysis

As the three experiments in the previous section show, max-margin based sample performs better than the unsupervised clustering (GMM) based technique, which in turn performs better than the other baseline techniques. In this section we investigate why the *MaxMargin classification* based technique outperforms other sampling techniques.

Our first hypothesis is based on the observation that *MaxMargin classification* based sample consists of higher number of incorrectly classified data items when compared to all other sampling techniques. Table IV shows the number of correctly and incorrectly classified data items in a 5% sample on for both datasets. A 5% sample, *MaxMargin classification* based sample has 79% and 85% of the incorrectly classified data items for the MNIST and GZoo2 datasets

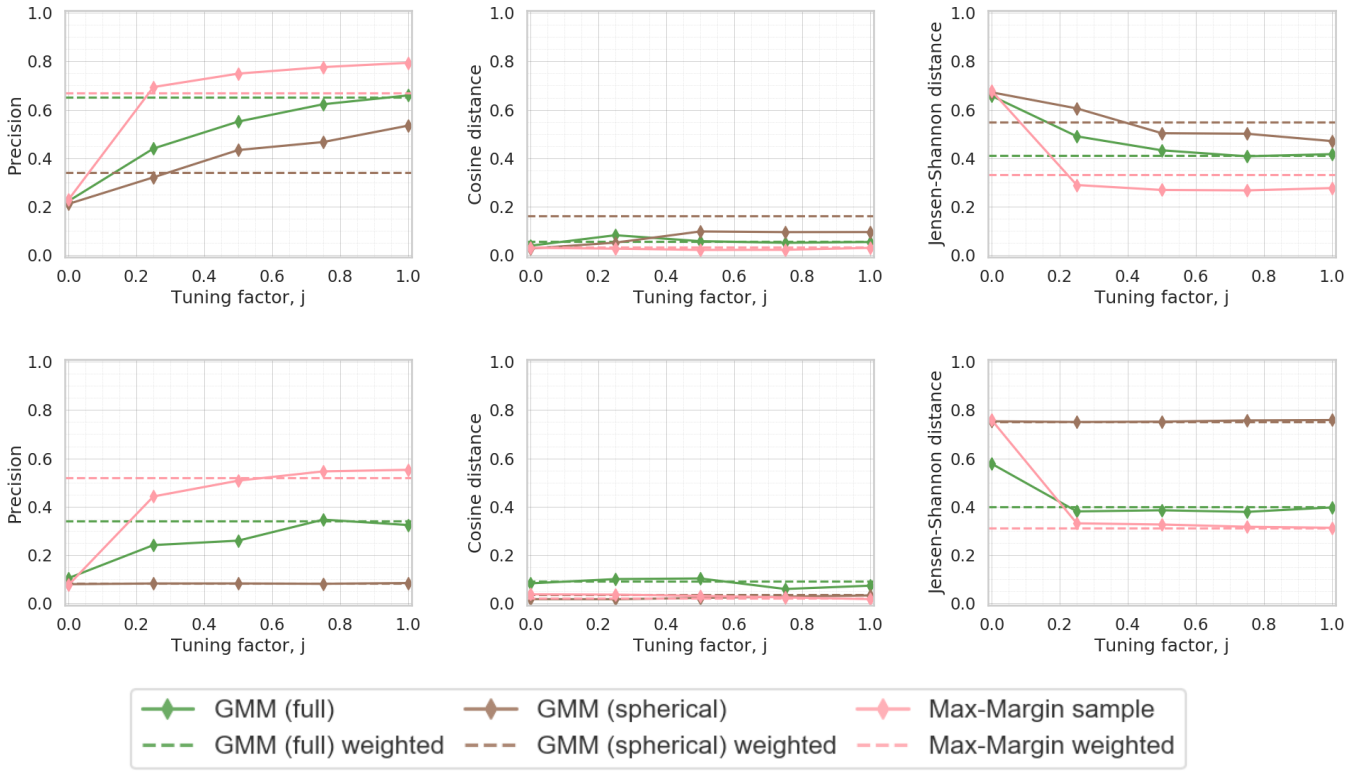


Fig. 7: Impact of tuning factor j (x-axis) on metrics for MaxMargin and GMM based sampling strategies. From top row MNIST, bottom row Galazy Zoo2, from left to right columns S1, S2, and S3.

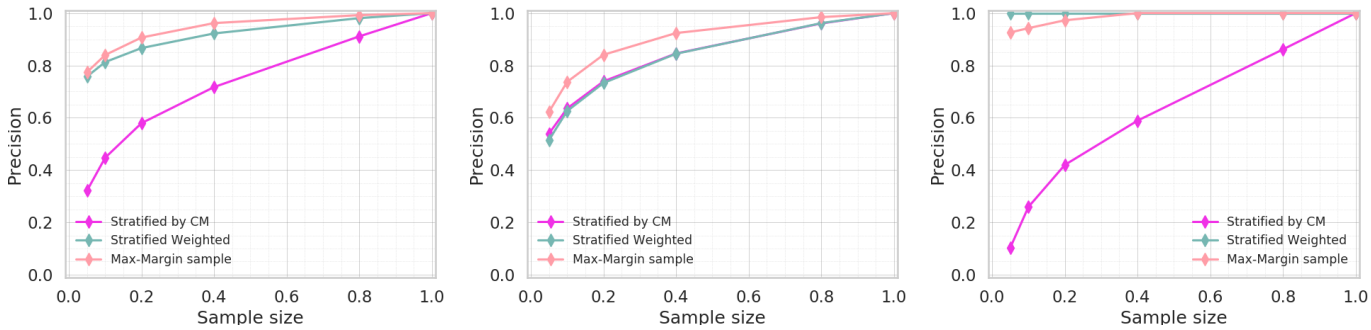


Fig. 8: Precision for query set S1 for MNIST. From left: precision on the query set, metrics for a single query for correctly, and incorrectly classified data items.

respectively. In comparison, *stratified by CM* sample has 10% of the incorrectly classified data items. We hypothesize that a sample with larger number of incorrectly classified data items outperforms samples with fewer incorrectly classified data items. To evaluate this hypothesis we create another sample *stratified weighted*, where we over-sample incorrectly classified data items. *Stratified weighted* sample is based on *stratified by CM* sample with one difference, instead of sampling uniformly from correctly and incorrectly classified data instances for each class, we pick 50% of the data items from incorrectly classified data instances and rest of the data items from the correctly classified data instances for each class. *Stratified weighted* and *MaxMargin classification* based samples now have comparable number of incorrectly classified data items, while *stratified by cm* sample always exceeds the number of correctly classified data items when compared to *MaxMargin classification* based sample. Figure 8 illustrates the performance of *stratified weighted*, *stratified by CM* and *MaxMargin classification* based samples on query set S1 on MNIST dataset.

Examining results of this new sampling technique, we observe that *MaxMargin classification* based sample outperforms *stratified weighted* sample on the query set S1 (first column, Figure 8) despite having the same number of incorrectly

classified data items. To understand this further we examine the performance of the three sampling techniques on a specific query for correctly and incorrectly classified data items (second and third column of Figure 8 respectively). Figure 8 shows results for correctly and incorrectly classified data items for MNIST dataset. This shows that while the *stratified weighted* and *MaxMargin classification* based samples have similar performance for incorrectly classified data items, for correctly classified data *MaxMargin classification* based sample outperforms *stratified weighted* sample. Additionally *MaxMargin classification* based sample outperforms *stratified by CM*, which has more correctly classified data items in the sample. This implies that compared to other sampling techniques *MaxMargin classification* based sample is able to capture the diversity of activation values for neurons, and overall outperforms both *stratified by CM* and *stratified weighted* samples. This is notable, because the *MaxMargin classification* sampling technique only utilizes the distance from the decision boundary to pick the sample rather than the actual classification result (whether the data item was correctly or incorrectly classified). Both of the models we examine are highly accurate, thus selecting 50% of the incorrect instances results in selecting all of the incorrectly classified data items. If the underlying models were not highly accurate, simply selecting a larger number of incorrectly classified data instances at random would perform similar to the graphs of correctly classified data items, where *MaxMargin classification* based sample outperforms the *stratified weighted* sample. It is possible that by carefully selecting a sample such that it consists of specific number of correct and incorrectly classified items for each class for models where the model accuracy was low, we could create a sample that performs at par with the *MaxMargin classification* based sample. However such a sample would need careful parameter selection for every DL model being diagnosed, the advantage of *MaxMargin classification* based sample creation is that it enables the selection of sample automatically in a principled manner without tuning requirement for every DL model being diagnosed.

However, *MaxMargin classification* has a parameter, the tuning factor. We examine the impact of tuning factor on the performance on query sets in order to validate that selecting a larger proportion of the sample from the decision boundary results in superior performance of the *MaxMargin classification* based sample and determine if there is a suitable default value of this tunign factor. This analysis will enable selection of a default value of the tuning factor. We examine the impact of tuning factor on all three query sets. As previously described, *MaxMargin classification* based sample is created by selecting items from the two ends of the list of data items sorted by distance from the decision boundary. Tuning factor of 0.0 implies all of the items in the sample are from the largest distance to decision boundary end of the list and, tuning factor of 1.0 implies all of the items in the sample are from the smallest distance to the boundary end of the list. Finally, tuning factor 0.5 implies half of the items in the sample are farthest from the boundary and half from closest to the boundary. Figure 7 shows the impact of tuning factor on all three query sets on both datasets, and Figure 10 shows the impact of tuning factor on single query (on a single layer and single class) for correctly (top row) and incorrectly (bottom row) classified data items.

Figure 7 shows that sample based on items closest to the boundary (tuning factor:1.0), outperform the sample based on items farthest (tuning factor:0.0) from the boundary for all the query sets on both datasets. Notably, this is true for all three query sets. The detailed query results in Figure 10 show the impact of tuning factor on a single query (on one layer, for one class) from each of the three query sets on the MNIST dataset. Figure 10 shows that a tuning factor of 0.0 results in very poor performance on incorrectly classified data items (bottom row), and tuning factors 1.0 and 0.5 perform similarly for incorrectly classified data items. Additionally, all three queries for correctly classified data items (top row, Figure 10) also show lower performance of sample with tuning factor 0.0. For the correctly classified data items top-k query tuning factor of 1.0 provides the best performance. However, for the average neuron activation values query with tuning factor of 0.75 demonstrates a performance improvement between 22% - 68% over tuning factor:0.0 and an improvement between 20% and 63.3% over tuning factor:1.0. Similarly, the maximally activated neuron distribution query with a tuning factor of 0.75 demonstrates a performance improvements between 2% to 45% over tuning factor:0 and, an improvement of between 5% to 14% over tuning factor:1.0.

This supports the hypothesis that selecting the sample from data items closest to the decision boundary captures the diversity of the neuron activation values better than selecting the sample with data items farthest from the decision boundary. We can also see that choice of tuning factor would be influenced by distribution of data items in the latent space. In the two dimensional (t-SNE) projection of the data in Figure 1 we can see that data items from each class are form compact clusters with clear separation between different clusters for MNIST when compared to galaxy zoo2 data. Thus, the sample farthest from the decision boundary would be a lot more homogeneous, i.e. have data items with similar important neurons for MNIST dataset rather than for the Galaxy Zoo2 dataset. Figure 7 supports this, the difference in the sample performance for tuning factor 0.0 and 1.0 is smaller for galaxy zoo2 dataset when compared to MNIST. Additionally, the performance of samples with tuning factor 0.0 and 1.0 converge at lower sample sizes for the galaxy zoo2 dataset when compared to MNIST. An overall of tuning factor between 0.5 and 1.0 performs best, an 0.7 is a suitable default value.

E. Query timing and Sample Creation Overhead

In this section we evaluate the query execution time (from python) as well as the time to create the sample. We measure query execution time for a subset of queries from Table II. Specifically we evaluate queries Q1 through Q5.

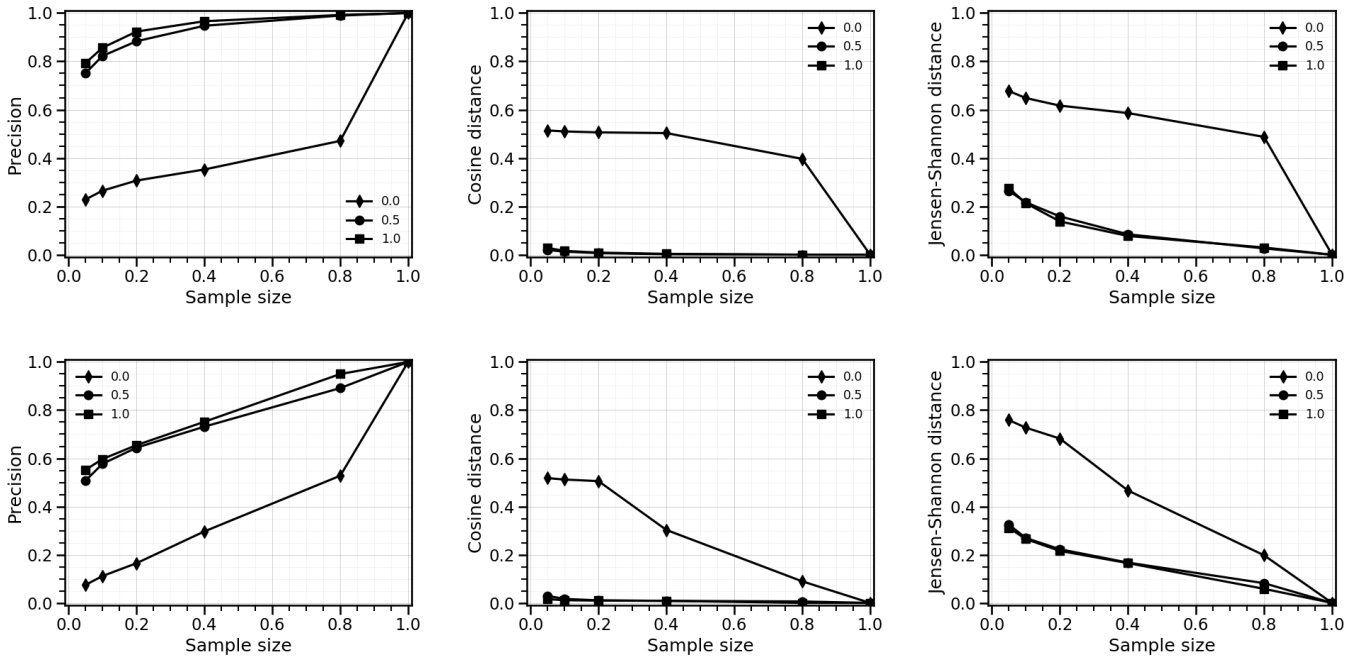


Fig. 9: Impact of tuning factor j on metrics for Max-Margin sample for S1, S2 and S3, by increasing sample size for both datasets. MNIST:top row; Galaxy Zoo2: bottom row.

In Figure 11 we show the execution time in seconds for these queries on all both data sets for 5%, 20% sample, and the full training dataset. While the query runtime trends seem very disparate, the improvements in runtime for both models are consistent. In the 20% sample we see $3.36-6.8\times$ improvement in query runtime, and for the 5% sample we see $10-30\times$ improvement in runtime. Q1 and Q2 have the longest runtimes on the full data set, as conv2, the layer for these two queries is the second convolutional layer in the models. The size of activations for the layer conv2, is $752.6MB$ for MNIST and $62GB$ for Galaxy Zoo2 on the full training data. While Q1 is looking for average activation for each neuron, Q2 is looking for average activation for incorrectly classified data points. To compute the results for Q2, we first join class label and predicted values with conv2 activations, filter to retrieve the incorrectly classified data points, and then compute the average for this group. For Q1, we need to load the activations for conv2 and compute average activations. Thus, when the data is large, join and filter operations required for Q2 make it take longer than Q1. This can be seen in the longer runtime for Q2 when compared Q1 for Galaxy Zoo2. Q4 computes the similarity between the logits. Logits are the un-normalized activations from the last layer in the model. For both data sets the time for the similarity computation is similar, ~ 0.05 seconds, rest of the time for the query is spend in loading activations. This time is proportional to size of the data, which is $752.6MB$ for MNIST and $8.GB$ for Galaxy Zoo2 for the entire training set.

Q5 is the fastest running query for MNIST and CIFAR-10 datasets. Q5 operates on the activations of last convolutional layer. Q3 operates on the activations last hidden layer which is *smaller* in size than the last convolutional layer, but Q3 takes longer to run than Q5 for these two datasets. This is because in addition to reading the data in Q3, we need to join the activations with class labels and predicted values, filter the result, and aggregate.

Q3 takes similar time for all three datasets. This is because the size of last hidden layer for the Galaxy Zoo2 data set is in similar range to MNIST as the size of this layer depends on the number of neurons and number of data points. Predictably, Q5 takes longer for Galaxy Zoo2 as the size of activations from the last convolutional layer is $10\times$ for MNIST. Above results demonstrate that running queries on a sample instead of the full data set results a savings of $3\times$ to $6.8\times$ for the 20% sample and between $10\times$ to $30\times$ on a 5% sample.

The queries above required all of the activations to be pre-generated. Thus, next we examine storage cost associated with storing and querying samples instead of the full data set. Unlike other sampling databases such as BlinkDB, our sampling technique offers storage savings in addition to query execution time savings. This is because creation of samples requires the activation values from the last hidden layer. Each of the sampling techniques requires as input the entire data sets representation in model space and as output returns the IDs of the data points in sample. Thus, creating a sample does not need all of the activations for all of the layers in the model to be generated. We only need to generate activation values for the last hidden layer. Figure 12 shows the storage saving for

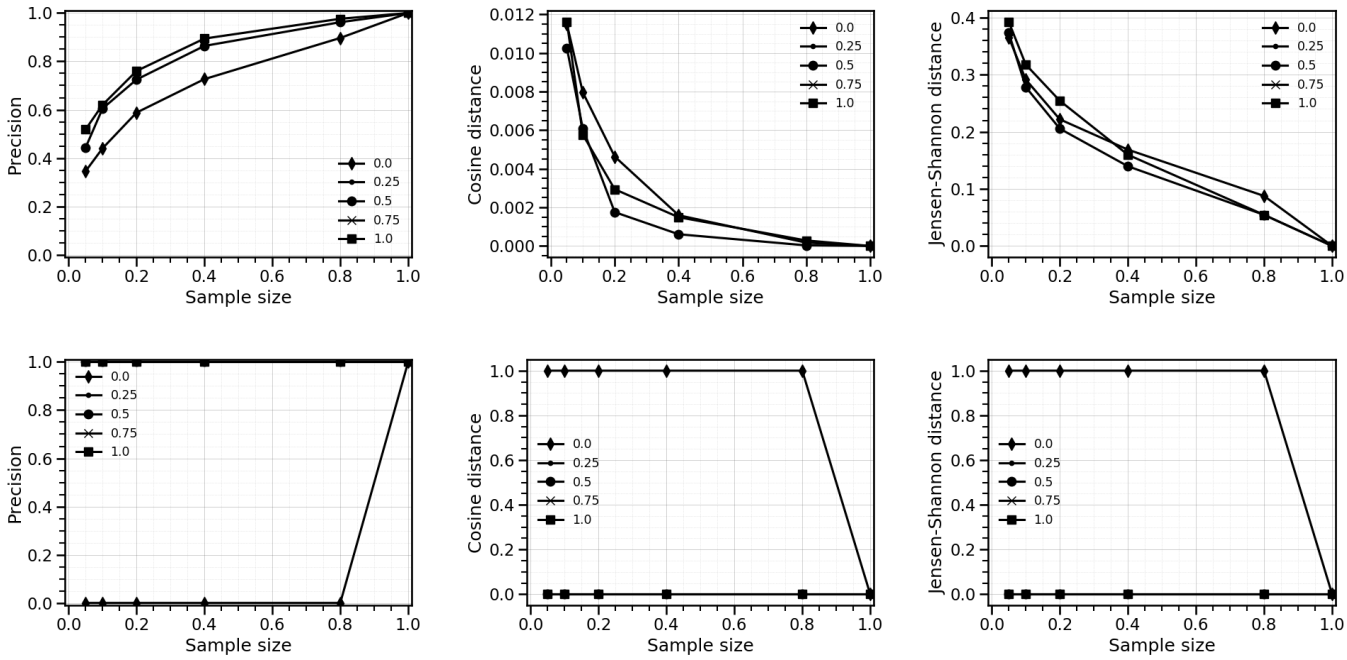


Fig. 10: Impact of tuning factor j on metrics for Max-Margin sample for single query from query sets S1, S2 and S3 for on specific correct (top row) and incorrect (bottom row) classified data items on MNIST dataset.

entire data from the three data sets. As expected the saving from a 5% sample are $\sim 20\times$, and from 20% sample are $\sim 5\times$. As seen in Section V depending upon the tolerance for error a 5% sample may be adequate, thus ML practitioners can expect upto $20\times$ reduction in storage footprint with our sampling technique.

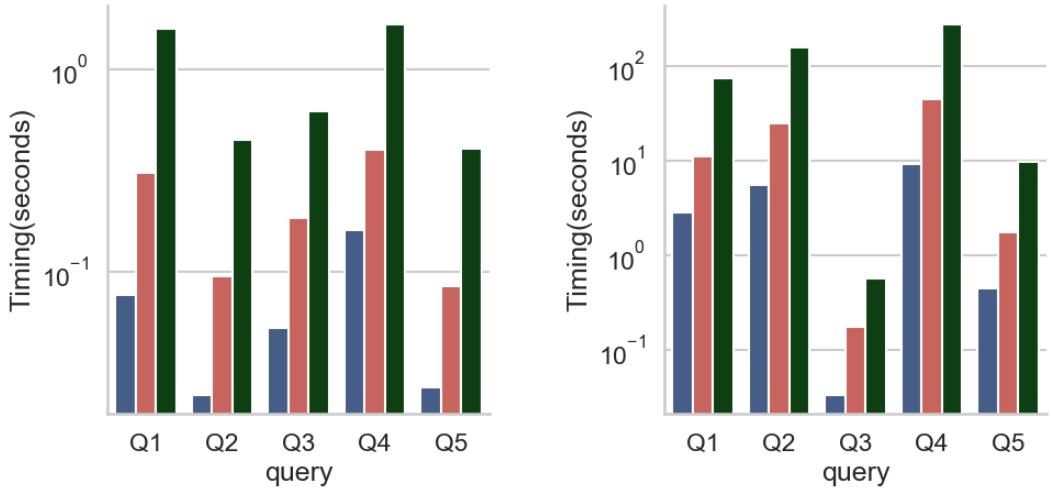
Finally, we examine the time it took to create these samples for baselines as well as for our sampling techniques. Figure 13 depicts the time required to generate a 5% sample for all sampling techniques on the Galaxy Zoo2 test dataset; note the log scale on the y-axis. Results for the MNIST dataset were similar and are not shown. The Galaxy Zoo2 test set had 2118 points, each point is a vector of size $[1, 64]$. Generating the uniform sample was the fastest as expected. Generating, *stratified by CM* sample, and *GMM* samples, required similar time, taking less than a second. Generating *MaxMargin classification* based sample required 1.5 seconds. Both VAS and EB-tree samples took three orders of magnitude more time. VAS is created with the interchange algorithm [37], each point in the dataset has to be added and one point evicted, by comparing proximity of the added point with each element of the existing sample. This is $O(K_2N)$ where K is the sample size and N is number of points in the dataset. For large datasets as in cases of ML, the time to create this sample was unacceptably long. Boundary stitching algorithm [55] is $O(NK)$. This was faster than the VAS but still took longer than our sampling technique.

VI. Related Work

Our work is related to three different categories of research; approximate query processing, model diagnosis systems, and model lifecycle management and tuning systems. We review work from each of these categories below.

1) Approximate query processing (APQ) and top-K queries: ([1], [3], [6], [18]) APQ is a well-studied area in databases and is an effective technique to deal with large-scale data. Algorithms for exact top-k queries are defined by the seminal work on the threshold algorithm (TA) [13], which require access to the indexed attribute(s) for a data set. Efficient processing of the top-k queries over samples is a challenging task [19]. Related work in this category includes top-k processing techniques that operate on deterministic data but report approximate answers in favor of performance. The approximate answers are usually associated with probabilistic guarantees; indicating how far they are from the exact answer. Algorithms presented in [50] are an approximate adaptation of TA where the approximate answers to the top-k query is associated with probabilistic guarantees. However, like TA this algorithm requires access to sorted attributes for the underlying data. Another approach to approximate top-k answers is considered in similarity search for multi-media databases [4]. This method uses a proximity measure to determine if a data region should be inspected. This utilizes the underlying data distribution rather than individual column value and in that sense is closer to our approach (i.e., instead of examining the underlying data, we utilize the latent space to create a sample).

2) Model diagnosis systems: ([5], [21], [22], [32], [53]) Model tracker [5] is one of the earliest systems for model diagnosis. It diagnoses models by tracking its performance using statistical measures, such as accuracy, AUC, etc. and does not



(a) MNIST

(b) Galaxy Zoo2

Fig. 11: Query timing for Q1 - Q5 from Table II for the 5% sample, 20% sample and entire dataset for both datasets (y-axis uses log scale).

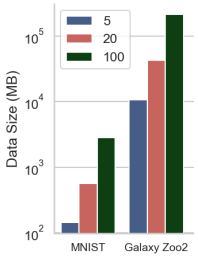


Fig. 12: Data size for full, 20%, 5% samples for both data sets.

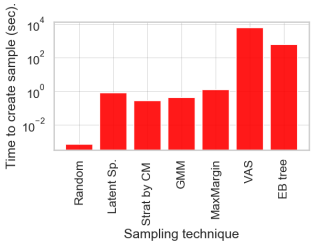


Fig. 13: Time to create a 5% sample for Galaxy Zoo2 dataset for all sampling techniques.

support model diagnosis for DL models. MLCube [22], one of the earlier visualization tool for model diagnosis visualizes data from pre-computed data cubes based on features from data and model results. The data-cubes utilized by this tool are based on less than 100 features and like Model tracker, it pre-dates the large scale of data that must be supported for DL model diagnosis. MISTIQUE [53] supports DL model diagnosis via examination of model activations, their primary approach is to reduce the storage footprint required by activations. MISTIQUE shares our goals of reducing query runtime for model diagnosis, but it uses a different approach, quantization and de-duplication to reduce the storage. Modelhub [32] supports model diagnosis by storing learned models and training logs with an approach that reduces storage footprint. Modelhub focuses on different artifacts, learned models and training logs, which they store and retrieve efficiently by introducing a model versioning system and a domain-specific language for searching through model space, solving a very different problem. DeepBase [41] supports model interpretabiltiy and diagnosis by providing a declarative abstraction to express and execute the generation and comparision of these artifacts. DeepBase relies on the ability to encapsulate model interpretbility questions as hypothesis functions (e.g., parts of speech tags and image captions). DeepBase, ModelHub and MISTIQUE could benefit by leveraging our sampling techniques for their systems. Finally, a variety of visualization tools [21], [29], [46], [49], [57] utilize activations and gradients to interpret and diagnose DL models. All of these tools would benefit from our sampling techniques, as sampling would help reduce the scale of data required to support model diagnosis. Activis [21], for instance selectively pre-computes values for nodes of interest to save computation and storage. Sampling techniques such as ours will enable ML practitioners using tools such as Activis to avoid making such compromises.

3) Model lifecycle management and model tuning: ([12], [33], [45], [52]) ModelDB [52] is a system for managing of ML models and pipelines. It provides versioning and metadata-based search and validation on models, simplifying the model building pipeline. MLflow [33] tracks experiments, packages the code to create reusable deployments and operationalizes the chosen models, addressing a very different aspect of

model lifecycle management compared to ModelDB. However, neither of these systems help manage, store, or query any DL model diagnosis artifacts. While MLflow supports storing and tracking arbitrary artifacts in a framework and implementation agnostic manner, it does not utilize information such as representation learned by the models to help with the selection of appropriate model. In addition custom code has to be provided for generating and querying these artifacts in MLflow. These tools do not support model diagnosis or interpretability as a primary goal, if they were to adopt model diagnosis as a goal our sampling technique could help with managing the size of data required.

VII. Conclusion and Future work

Deep learning models have become an indispensable tool for a wide range of tasks, such as image classification, object recognition, speech analysis, machine translation, and more. The task of diagnosis for these purportedly black-box models requires additional artifacts, such as activations. These additional artifacts must be generated, stored, and queried for each DL model being debugged. The addition of these artifacts, which can be up to three orders of magnitude larger than the input data size for each model being diagnosed, turns the process of building, diagnosing, and selecting DL models in to a large-scale data management challenge. In this work, we quantify DL diagnosis workload and present a novel sample creation technique that reduce the time and complexity required to accomplish these tasks.

The sampling technique we present in this paper focus on sampling input data points, e.g. rows from the relation of data points and activations. The ML literature supports the notion of reducing the number of neurons for which activations need to be calculated [29], [31] and queried. We would like to explore this avenue in future work. The sampling technique described in this paper works well with supervised learning models, i.e. DL models built with labeled data. In future work, we would like to explore our sampling technique and their efficacy for *unsupervised* DL models, such as generative models, autoregressive models, etc. [10] A large body of scientific data is unlabeled and requires unsupervised learning techniques, and extending our sampling technique in this direction could be beneficial to the scientific community working on newer data sets.

References

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *SIGMOD '99*.

[2] https://en.wikipedia.org/wiki/Activation_function.

[3] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *EuroSys '13*, 2013.

[4] G. Amato, F. Rabitti, P. Savino, and P. Zezula. Region proximity in metric spaces and its use for approximate similarity search. *ACM Transactions on Information Systems (TOIS)*, 21(2):192–227, 2003.

[5] S. Amershi, D. M. Chickering, S. M. Drucker, B. Lee, P. Y. Simard, and J. Suh. Modeltracker: Redesigning performance analysis tools for machine learning. In *CHI*, 2015.

[6] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD '03*, 2003.

[7] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition*, 2017.

[8] https://en.wikipedia.org/wiki/Cosine_similarity.

[9] https://data.galaxyzoo.org/gz_trees/gz_trees.html.

[10] https://deepmind.com/blog/unsupervised-learning/.

[11] H. Domínguez Sánchez, M. Huertas-Company, M. Bernardi, D. Tuccillo, and J. Fischer. Improving galaxy morphologies for sdss with deep learning. *Monthly Notices of the Royal Astronomical Society*, 2018.

[12] C. B. et.al. Towards interactive curation and automatic tuning of ml pipelines. In *DEEM*, 2018.

[13] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003.

[14] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 2008.

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[16] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5), Aug. 2018.

[17] https://www.zooniverse.org/projects/zookeeper/galaxy-zoo.

[18] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD '97, Proceedings*, 1997.

[19] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

[20] https://en.wikipedia.org/wiki/Jensen%E2%80%93Shannon_divergence.

[21] M. Kahng et al. Activis: Visual exploration of industry-scale deep neural network models. *IEEE TVCG*, 2018.

[22] M. Kahng, D. Fang, and D. H. P. Chau. Visual exploration of machine learning results using data cube analysis. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, page 1. ACM, 2016.

[23] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[24] https://en.wikipedia.org/wiki/Kernel_method.

[25] S. Kornblith, M. Norouzi, H. Lee, and G. E. Hinton. Similarity of neural network representations revisited. In *ICML '19*, 2019.

[26] S. Kornblith, M. Norouzi, H. Lee, and G. E. Hinton. Similarity of neural network representations revisited. In *ICML '19*, 2019.

[27] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[28] Y. Lecun. Gradient-based learning applied to document recognition. 1998.

[29] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[30] A. Mahendran et al. Visualizing deep convolutional neural networks using natural pre-images. *IJCV '16*, 2016.

[31] R. Maithra et al. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *NIPs*, 2017.

[32] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Modelhub: Deep learning lifecycle management. *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017.

[33] https://mlflow.org/.

[34] http://yann.lecun.com/exdb/mnist/.

[35] A. S. Morcos, M. Raghu, and S. Bengio. Insights on representational similarity in neural networks with canonical correlation. In *NeurIPS '18*, 2018.

[36] C. Olah et al. Feature visualization. *Distill*, 2017.

[37] Y. Park, M. J. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, 2016.

[38] J. Platt. Probabilistic outputs for svms and comparisons to regularized likehood methods, advances in large margin classifiers, 1999.

[39] https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning.

[40] http://skyserver.sdss.org/dr7.

[41] T. Sellam, K. Lin, I. Y. Huang, M. Yang, C. Vondrick, and E. Wu. Deepbase: Deep inspection of neural networks. In *SIGMOD '19, Amsterdam*, 2019.

[42] R. R. Selvaraju et al. Grad-cam: Visual explanations from deep networks via gradient-based localization. *ICCV'17*, 2018.

[43] K. Simonyan et al. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR'13*, 2013.

[44] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[45] E. Sparks et al. Automating model search for large scale machine learning. In *SoCC'15*, 2015.

[46] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Trans. Vis. Comput. Graph.*, 2018.

[47] M. Sundararajan et al. Axiomatic attribution for deep networks. In *ICML*, 2017.

[48] https://en.wikipedia.org/wiki/Support-vector_machine.

[49] http://www.tensorflow.org/.

[50] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 648–659. VLDB Endowment, 2004.

[51] L. van der Maaten and G. E. Hinton. Visualizing data using t-sne. In *Journal of Machine Learning Research 9, Nov*, 2008.

[52] M. Vartak. Modeldb: A system for machine learning model management. In *CIDR'17*, 2017.

[53] M. Vartak, J. M. F. da Trindade, S. Madden, and M. Zaharia. Mistique: A system to store and query model intermediates for model diagnosis. In *SIGMOD Conference*, 2018.

[54] H. Wu, C. Wang, J. Yin, K. Lu, and L. Zhu. Interpreting shared deep learning models via explicable boundary trees. *ArXiv*, abs/1709.03730, 2017.

[55] H. Wu, C. Wang, J. Yin, K. Lu, and L. Zhu. Sharing deep neural network models with interpretation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, 2018.

[56] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5(Aug):975–1005, 2004.

[57] J. Yosinski, J. Clune, A. M. Nguyen, T. J. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *CoRR*, 2015.

[58] M. Zeiler et al. Visualizing and understanding convolutional networks. In *ECCV '14*, 2014.