Using Deep Learning for Big Spatial Data Partitioning

TIN VU, University of California, Riverside
ALBERTO BELUSSI and SARA MIGLIORINI, University of Verona
AHMED ELDWAY, University of California, Riverside

This article explores the use of deep learning to choose an appropriate spatial partitioning technique for big data. The exponential increase in the volumes of spatial datasets resulted in the development of big spatial data frameworks. These systems need to partition the data across machines to be able to scale out the computation. Unfortunately, there is no current method to automatically choose an appropriate partitioning technique based on the input data distribution.

This article addresses this problem by using deep learning to train a model that captures the relationship between the data distribution and the quality of the partitioning techniques. We propose a solution that runs in two phases, training and application. The offline training phase generates synthetic data based on diverse distributions, partitions them using six different partitioning techniques, and measures their quality using four quality metrics. At the same time, it summarizes the datasets using a histogram and well-designed skewness measures. The data summaries and the quality metrics are then use to train a deep learning model. The second phase uses this model to predict the best partitioning technique given a new dataset that needs to be partitioned. We run an extensive experimental evaluation on big spatial data, and we experimentally show the applicability of the proposed technique. We show that the proposed model outperforms the baseline method in terms of accuracy for choosing the best partitioning technique by only analyzing the summary of the datasets.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Information systems** → **Database management system engines**;

Additional Key Words and Phrases: Spatial partitioning, deep learning, skewed data, data synopsis

ACM Reference format:
Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldway. 2020. Using Deep Learning for Big Spatial Data Partitioning. *ACM Trans. Spatial Algorithms Syst.* 7, 1, Article 3 (August 2020), 37 pages.
https://doi.org/10.1145/3402126

1 INTRODUCTION

In recent years, there has been a notable increase in the amount of spatial data produced by IoT sensors, social networks, and autonomous vehicles, among others. This led to many research

This work was partially supported by the Italian National Group for Scientific Computation (GNCS-INDAM) and by "Progetto di Eccellenza" of the Computer Science Dept., Univ. of Verona, Italy. This work was also supported in part by the National Science Foundation (NSF) under grants IIS-1838222 and CNS-1924694 and by the USDA National Institute of Food and Agriculture, AFRI award number 2019-67022-29696.
Authors' addresses: T. Vu and A. Eldway, University of California, Riverside, 900 University Ave, Riverside, CA 92521, USA; emails: tin.vu@email.ucr.edu, eldawy@ucr.edu; A. Belussi and S. Migliorini, University of Verona, Strada Le Grazie 15, 37134 Verona VR, Italy; emails: {alberto.belussi, sara.migliorini}@univr.it.

2374-0353/2020/08-ART3 $15.00
https://doi.org/10.1145/3402126

Table 1. Execution of the DJ in SpatialHadoop with Different Kinds of Indexes (i.e., Gr = regular grid, Qt = Quadtree, Rt = R-tree) and Different Distributions of the Datasets (i.e., Uni = uniform distribution, Skw = skewed distribution)

Dataset distribution	Dataset index	Tot. time (mills)	Map tasks		
			# tasks	AVG time (millis)	%RSD time
Uni/Uni	**Gr/Gr**	**145,307**	37	15,833	4%
Uni/Uni	Gr/Qt	150,458	51	18,902	9%
Uni/Uni	Gr/Rt	147,646	54	16,231	7%
Uni/Skw	Gr/Gr	125,327	33	22,710	90%
Uni/Skw	Gr/Qt	96,001	52	11,209	50%
Uni/Skw	**Gr/Rt**	**40,205**	21	18,087	28%

tasks is the total number of map tasks, AVG time is the average time for a map task, and %RSD is the relative standard deviation for the running time of map tasks.

efforts for developing *big spatial data* frameworks that are able to absorb and process these huge amounts of data such as SpatialHadoop [13], Simba [38], GeoSpark [39], and others [14, 28, 32]. Regardless of their internal architecture, all these systems have a common and necessary first step, that is, *spatial data partitioning*. These systems *scale out* by partitioning the data across machines and then processing these partitions in parallel. However, there is no single partitioning technique that all the systems agree on. Rather, most of these systems provide a wide range of spatial partitioning techniques and it is up to the user to choose an appropriate one. Past studies showed that the spatial partitioning approach is critical to the performance of many spatial analytic operations such as indexing [11], computational geometry [12], visualization [16], spatial joins [13], kNN joins [24], and others.

Choosing an appropriate spatial partitioning technique is a very challenging and complicated problem for two reasons. First, the efficiency of these partitioning techniques rely on the characteristics and distribution of the dataset, e.g., uniform Vs skewed data, points Vs rectangles, or clustered Vs scattered data. Second, the requirements of the analytic operations play a huge role in choosing a partitioning technique, e.g., maximize load balancing, minimize partition overlap, or prefer square-like partitions. Recent studies provided both theoretical [6, 8] and experimental [11] evaluations of several partitioning techniques for big spatial data and highlighted the complexity of choosing one technique over the others. As new partitioning techniques are developed [35], the problem becomes even more complex.

To illustrate the complexity of the problem, Table 1 shows the result of the execution in SpatialHadoop of the Distributed Join (DJ) [7, 15] applied to two synthetic datasets, where the first one is uniformly distributed (i.e., "*Uni*") and partitioned using a regular grid (i.e., "*Gr*"), while the second one varies from a uniform (i.e., "*Uni*") to a skewed (i.e., "*Skw*") distribution and has been partitioned using different techniques, namely regular grid (i.e., "*Gr*"), Quad-tree (i.e., "*Qt*"), and R-tree (i.e., "*Rt*"). Interestingly, when both datasets are uniformly distributed, the response time of the DJ is the best with the uniform grid partitioning with Rt and Qt coming as close second and third. However, when a skewed distributed dataset (*Skw*) is considered, then the differences are significant, and in this particular case are in favor of the R-tree-based partitioning technique. This is due mainly to the fact that when the distribution is skewed the partitioning of the geometries based on a regular grid does not produce balanced splits, while the Quad-tree and the R-tree-based partitioning techniques perform better and produce more balanced splits. This is evident from columns 4, 5, and 6

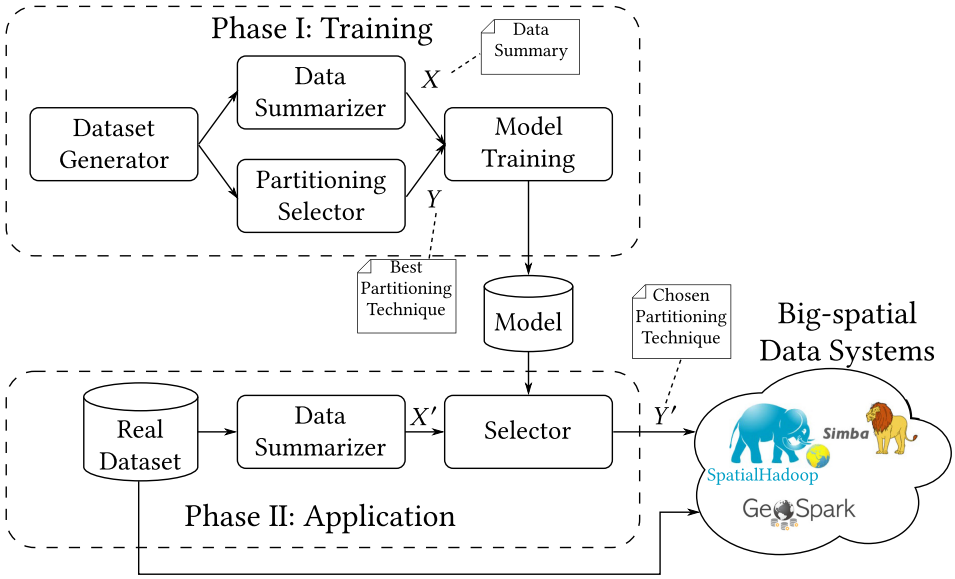


Fig. 1. The workflow of the proposed solution.

of Table 1, which report the characteristics of the map tasks in the different cases. In particular, column 4 contains the number of instantiated map tasks (which depends on the pair of intersecting partitions from both datasets), column 5 reports the average time taken by a map task, and column 6 shows the relative standard deviation of the execution time of the map tasks w.r.t. to their mean signifying the load balance. It is clear that balancing the cost of the single map tasks is crucial for the total cost of the MapReduce job, in particular when the implemented operation is performed primarily in the map phase.

The aim of this article is to define a new mechanism for choosing the most appropriate partitioning technique for a given dataset. There are three design goals for the proposed work: (1) ability to make a decision based on parameters that can be computed quickly, (2) support arbitrarily many partitioning techniques, and (3) provide different choice criteria based on the requirements of the analytic operation and the user preferences.

To achieve the three goals mentioned above, we propose the framework illustrated in Figure 1. The framework works in two main phases, namely, *training* and *application* phases. In the training phase, we build the *partition selection model* that is able to choose an appropriate index for any given dataset. This phase is executed as an offline phase, and it consists of the following four components.

- **Dataset Generator:** This component generates many diverse synthetic datasets that are used to train the model. This step is important for deep learning that needs a very large training set that catches as many input features as possible.
- **Data Summarizer:** This component takes every input dataset and computes a set of descriptors that summarizes the dataset and catches its details. This step transforms the variable-size input dataset to a fixed-size feature vector X that the deep learning algorithm can process. This article considers two summarization techniques, *fractal-based* techniques, which utilize skewness measures that are developed by experts, and a simple *histogram* that represents a detailed density map.

- **Partitioning Selector:** This component assesses the quality of all supported spatial partitioning techniques to choose the best one. It evaluates the performance of all the available partitioning techniques using a set of standard quality metrics and generates a label Y that contains the best partitioning technique for each quality metric. The deep learning can use the pair (X, Y) for training the model.
- **Model Training:** This last step takes the feature vector X and the label vector Y and uses deep learning to build a model that can estimate the performance vector Y given the features X.

The second phase, the *application phase*, uses the model produced by Phase I and applies it on a new (real) dataset, provided by the user, and chooses the most appropriate partitioning technique for it. This phase first computes the feature vector X' exactly as in Phase I but on the real dataset. Then applies the model M on the vector X' to produce an estimated performance vector Y' that encodes the most appropriate partitioning technique. The chosen technique, taking into account also the user requirements (i.e., which operation she/he needs to apply), can then be passed to any big-spatial data system, e.g., SpatialHadoop or GeoSpark, for actual data partitioning and analysis.

In this article, we build a prototype for the proposed system using six different partitioning techniques, five quality metrics, and two data summarization techniques. The first summarization technique uses a few well-crafted skewness measures for spatial data including box counting [6, 8] and Moran's Index [25]. The second summarization technique uses a simple histogram for the entire dataset that represents a details density map but could be harder to use by the machine learning component due to their big size. One of the goals of this article is to study which summarization technique works better for this problem. In other words, can the deep learning technique extract its own skewness measures from the histogram that outperforms the ones developed by the experts? We test the proposed framework using both synthetic and real big datasets to show the effectiveness of the proposed framework. The initial experiments show up to 90% accuracy with synthetic data and 80% with real data.

In summary, the contributions of the article are listed and presented hereby.

(1) **Training set generation:** Deep learning model require a sufficiently large and representative training set. In the considered context where the problem to address is to choose the most suitable partitioning technique for a given spatial dataset of unknown distribution, no training set is available (unlike the image classification problem where huge repositories are freely available on Internet). So the first contribution of this article is to propose an approach for generating a training set addressing this kind of problems, and this includes a set of algorithms for producing the training set in a reasonable amount of time. In particular, the application that generates the training set has been implemented in Spark.

(2) **Feature extraction:** Once a training set is generated, we need to decide the features that should be extracted from the dataset to use as input to the machine learning model. There is an agreement that the distribution of the dataset is the key feature for choosing the best partitioning technique but the question is as follows: Which descriptors should be chosen? Which statistical descriptor is the best one for supporting the choice of a correct partitioning technique? To answer this question, this article proposes two techniques. The first technique extracts an ensemble of carefully selected skewness measures that have been shown to catch several important features of spatial data including box-counting [6, 8] and Moran's Index. This techniques resembles classical image processing techniques that extracts manually designed image features. The second technique uses the dataset histogram as one big feature and let the modern deep learning

method extracts its own features from the histogram. We show in this article that this method is easier to implement, since it avoids hand-picking the skewness measures and, thanks to deep learning, can provide very high accuracy.

(3) **Experimental evaluation:** Finally, as third contributions we configure, train and test a Neural Network proving that the proposed idea is feasible. In the experiments we use a considerable amount of synthetic datasets with different distributions and some real huge datasets. The results support our intuition that the histograms can be a good choice for addressing the optimization issue regarding data partitioning.

The rest of this article is organized as follows. Section 2 formalizes the problem. Section 3 describes the training phase that builds the partition selection model. Section 4 describes the application phase applying the model to real datasets provided by the user. Section 5 provides an extensive experimental evaluation of the proposed system using real datasets. Section 6 describes the related work. Finally, Section 7 concludes the article.

2 PROBLEM DEFINITION

The problem that this article addresses is, given a spatial dataset, how to choose the best partitioning technique that will provide the best performance. Considering the case study shown in Table 1, it is evident that this is an important yet challenging problem given the complexity of big spatial datasets. In addition, the objectives of the spatial partitioning vary by the spatial operation that will be applied and the requirements of the system that applies this operation. For example, in selection and join operations, it could be desired to minimize the total area or total margin of the partitions [3, 4, 11]. However, for computational geometry operations [12], minimizing or eliminating the overlap between partitions could be more beneficial. For scanning and aggregate operations, load balance (i.e., minimize the variance) could be of a high advantage to minimize the straggler effect. This section aims at clearly defining the problem that includes how to identify the *best* partitioning technique.

Definition 2.1 (Feature (f)) A spatial feature f represents a record that contains a geometry g and a set of non-spatial attributes $A = \{a_i\}$. The minimum bounding rectangle $f.MBR$ is the smallest orthogonal rectangle that encloses the geometry g. The size $f.s$ is the total size of the feature representation, i.e., geometry plus attributes, in bytes. In this article, we do not process the actual geometry or attributes, rather, we only consider the MBR and size. A feature is also referred to as a *record* following the database terminology.

Definition 2.2 (Partition (P)). A spatial partition $P = \{f_1, \ldots, f_m\}$ is a set of spatial features that are stored in the same file block(s). The MBR, size, total number of blocks and average cardinality of blocks of the partition are defined as:

$$P.MBR = MBR\left(\bigcup_{f \in P} f.g\right) = MBR\left(\bigcup_{f \in P} f.mbr\right)$$

$$P.s = \sum_{f \in P} f.s$$

$$P.blocks = \lceil P.s/B \rceil$$

$$P.card = \frac{\mid P \mid}{P.blocks},$$

where B is the block size of the file system that has a default value of 128 MB in HDFS.

Any partitioning technique aims at producing partitions having at most one block; however, in practice the application of a technique to a real dataset D might produce also partitions containing more than one block, due to the particular distribution of the features of D in the reference space.

Definition 2.3 (Partitioning Technique (PT)). A partitioning technique ($PT : D \rightarrow \mathcal{P}$) is a function that can be applied to a dataset $D = \{f_i\}$ to produce a set of partitions $\mathcal{P} = \{P_k\}$ such that each feature f_i is assigned to at least one partition, i.e., $\bigcup_{P_k \in \mathcal{P}} P_k = D$.

Definition 2.4 (Quality Metric (QM)). The quality metric ($QM : \mathcal{P} \rightarrow \mathbb{R}$) is a function that is applied to a set $\mathcal{P}$ of partitions to quantify its quality as real number $qm \in \mathbb{R}$, e.g., the total area of partitions or standard deviation of the partition sizes or a combination of them.

Notice that the quality metric to be chosen might depend on the user requirements, i.e., the requested operation. Moreover, the quality metrics used in this article are better when having lower values, e.g., total area or total margin. However, there exist other quality metrics for which the higher the value the better, e.g., disk utilization. The approach proposed in this article can handle both types of quality metrics.

Next, we define the main problem that we address in this article.

Definition 2.5 (Partitioning Selection Problem (PSP)). Given a spatial dataset D, a set of partitioning techniques $PT = \{PT_1, \ldots, PT_n\}$, and a quality metric QM, choose the best partitioning technique PT_i that will minimize/maximize the quality metric QM when applied to the dataset D.

A naïve solution to the PSP problem is to apply all partitioning techniques to the big dataset and then compute the quality metric for all the resulting partitions and choose the best one. However, since the big spatial data frameworks deal with peta bytes of data, it is not feasible or effective to apply all possible partitioning techniques.

This article proposes a solution to this problem through a framework that uses deep learning to predict the best partitioning technique based on a history of how all partitioning techniques behave with datasets that are similar to the input dataset D. At a very high level, the proposed framework works in two phases: training and application. The training phase looks at a huge number of reference datasets and their quality when partitioned with all the available partitioning techniques. Then, it builds a small model M that captures this complicated relationship. The application phase takes a new dataset D and applies that model on D to choose a partitioning technique that is expected to be the best. This entails the following *challenging problems* that we address in this article.

- **Dataset generation:** How should we generate large and diverse reference datasets that can be used for training? These datasets should capture as many aspects of the partitioning techniques as possible. They should also simulate real datasets so that the generated model can be used with real data. We address this problem by surveying a large number of synthetic data distributions used in literature and choosing a set of representative distributions that are close to real datasets. Then, we generate a large number of datasets for each distribution by varying its parameters. Finally, we combine the generated datasets to generate more compound distributions that cannot be represented by a single distribution. This process has been done with the support of our open-source spatial data generator [36].
- **Dataset similarity:** One of the biggest problems is how to measure the similarity between different datasets including real datasets that are only available in the second phase. We evaluate and contrast two directions. The first direction uses some skewness measures defined by the experts such as box counting [6] and Moran's index [25]. The second direction uses a simple uniform histogram that is easier to compute but of a much larger size. The

second option is particularly intriguing to use with deep learning as the histogram looks like an image that deep learning is particularly good at.

- **Performance evaluation:** Given a dataset D, a set of partitioning techniques PT, and a set of quality metrics QM, how to measure all the quality metrics for all the partitioning techniques on the dataset D to be able to identify the best one for training purpose? We address this problem by proposing a distributed Spark-based algorithm that is able to generate the partitions $\mathcal{P}$ for all partitioning techniques as one distributed job without really having to partition the actual features of D. This technique allows us to generate a large number of reference datasets in a short time to improve the accuracy of the model during the training phase.
- **Model training:** Given the reference datasets and their corresponding quality measures, how to build a model that captures this complicated relationship? To address this problem, we use deep learning to build such a model and explain in this article how we choose the parameters for this model and do the model training.

3 TRAINING PHASE

The training phase is responsible of building the machine learning model M that can choose the best partitioning technique for a dataset D. This phase works in four steps. (1) Generate a set of reference datasets to use as training set. (2) Summarize each training dataset into a fixed-size vector that is used for training. (3) Compute all quality metrics for each dataset and label each dataset with the best partitioning technique for each quality metric. (4) Apply deep learning to learn the relationship between the data summary and the best technique. Details of the four steps are provided below.

3.1 Training Set Generation

This section describes the distributions of the synthetic datasets that we use for model training. Different distributions of geometries in the reference space produce different behavior of the partitioning techniques, which provide very different subdivisions of the features in the resulting partitions. Figure 2 illustrates an example with four datasets: a uniformly distributed set of rectangles (*Uniform distr.*), a set of rectangles distributed around the diagonal of the reference space (*Diagonal line*), a set of rectangles distributed around the lower left and upper right corners of the reference space (*Double cluster*), and a real dataset containing the primary roads of the USA (*Primary roads*). Three partitioning techniques have been applied: regular grid, QuadTree, and RTree to all the datasets. The resulting partitions are shown by drawing the boundary of their MBRs on top of the datasets plots. Notice that the MBRs produced by different techniques are very different from each other. Thus, the dataset distribution is a vital characteristic for deciding the correct partitioning technique. To build an effective training set, it is crucial to generate datasets with different distribution, in particular with different kind of skewed distributions.

For all datasets, two common parameters are set, the reference space (a bounding rectangle of the input space), and the total size. In addition, each distribution can have some additional parameters that control the dataset generation. In particular, we consider the following dataset distributions exemplified in Figure 3:

- Uniform distribution: The dataset geometries are uniformly distributed inside the reference space (Figure 3(a)). A parameter s is adjusted to represent the maximum side length of each rectangle. This distribution models real datasets that are uniformly distributed, e.g., houses in suburbs.

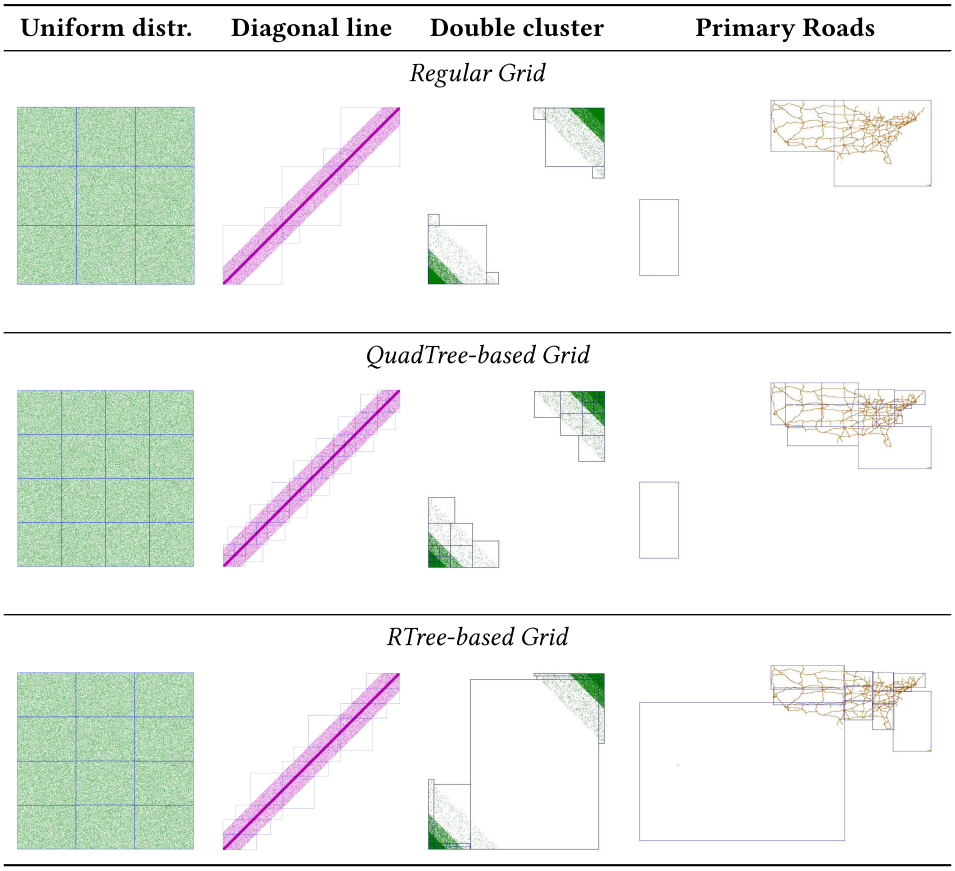


Fig. 2. Partitions produced by applying different techniques (first row: regular grid; second row: QuadTree-based grid; third row: RTree-based grid) for both synthetic and real datasets.

- Linear distribution: The dataset geometries are all located very close to a line, namely they are uniformly distributed inside a small buffer around it (Figure 3(b)). The training set considers as reference line both the main diagonal of the reference space, and about 100 possible rotations of it. This distribution can be customized by setting the maximum side length of a rectangle (s) and the size of the buffer (b). This distribution can represent data that are centered around a line, e.g., shops along a highway or houses along a river.
- Diagonal distribution: The dataset geometries are located around a line with a normal distribution. More specifically, the concentration of the geometries decreases as the distance from the main line increases (Figure 3(c)). In generating the various datasets, the percentage of geometries concentrated around the line and the dimension of overall buffer are changed. Moreover, beside to the main diagonal, we consider as reference line also about 100 possible rotations of it. This distribution can model data around a linear region such as river banks.
- Parcel distribution: This dataset is generated by recursively splitting the reference space by horizontal and vertical lines. After that, each resulting rectangle is randomized by slightly changing its size (Figure 3(d)). The parameter r represents the randomization factor as a percentage of the rectangle size. Parcel distribution can model some real datasets such as farm lands and green areas that cover a large region with slight or no overlap.

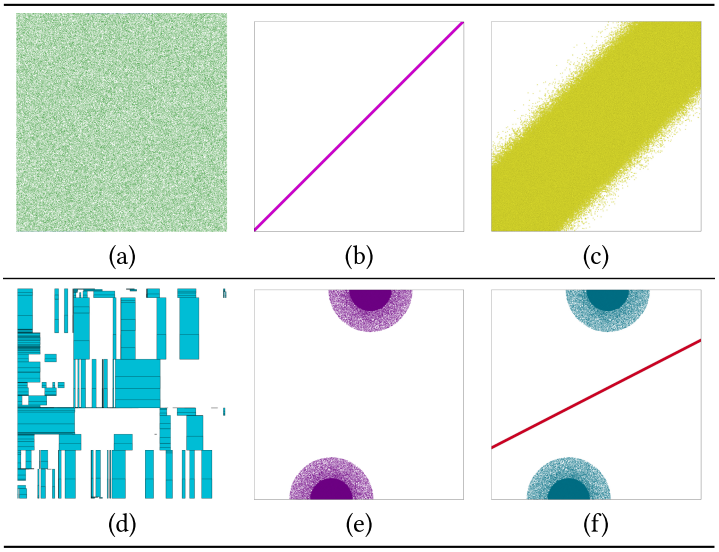


Fig. 3. Example of distributions contained in the training set.

- Cluster distribution: The dataset geometries are located around two main kernels. In particular, the majority of geometries are placed inside a smaller buffer around one of the two kernels, while the remaining ones are inside of a bigger buffer (Figure 3(e)). To produce the various datasets the percentage of closer geometries and the dimension of the two buffers are changed, as well as their position. The parameters for this distribution consist of the locations and sizes of the two centers. Cluster distributions can represent urban areas that are centered around big cities.
- Combinations of two of the previous distributions: several combinations of the above distributions have been produced. Figure 3(f) shows an example of combination between a cluster and linear distribution. These combinations allow for producing more complicated datasets that cannot be represented with a single distribution.

In generating the synthetic datasets, also the length of the rectangle sides have been changed to obtain datasets with small and big rectangles. A separate group of datasets have been generated for representing the MBRs produced by linear networks or similar real data where oblong rectangles are very frequent. Some snapshots of diagonal datasets extracted from the generated data are shown in Figure 4. This method is also applied to the other distributions to vary the shapes of the rectangles.

The experiments section provides the details of the parameters and sizes of the synthetic datasets that we use in our experimental evaluation.

3.2 Dataset Summarization

This part describes how we summarize the big and variable-size datasets into a fixed-size vector that catches their characteristics and can be used as an input to the deep learning model. We consider two types of summarization techniques, fractal-based and histogram-based techniques. The fractal-based technique is inspired by sophisticated skewness measures developed by research experts in literature, e.g., box-counting [5] and Moran's-Index [25]. Since inspired by experts, these skewness measures are supposed to make an effective summary of the input dataset. However,

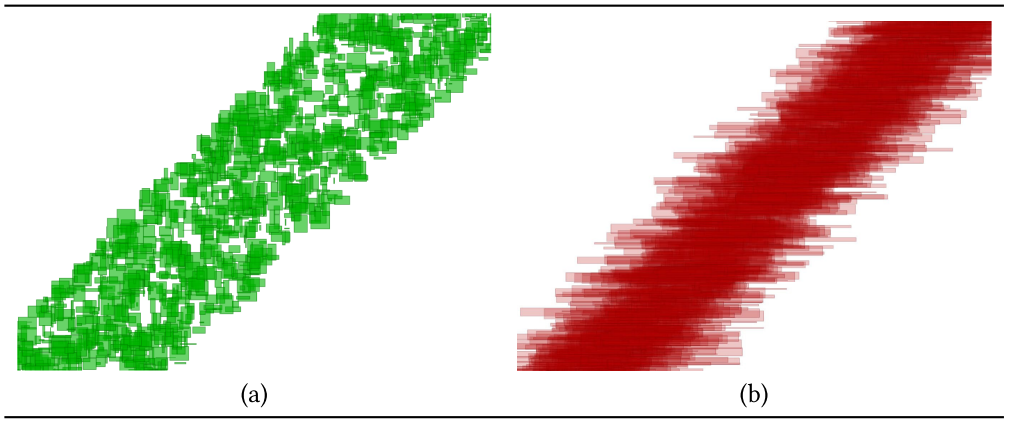


Fig. 4. Example of rectangles contained in the training set. (a) Regular rectangles of different sizes and (b) oblong rectangles of different sizes.

the *histogram* technique is basically a uniform histogram that is much bigger in terms of representation size but might be able to catch more details about the dataset. The research question that we address in this article is as follows: Can the machine use deep learning to come up with its own skewness measures based on the histogram that outperforms the fractal-based techniques developed by experts?

Considering the case study shown in Table 1, it is clear that an easy and efficient way for evaluating the skewness of a spatial dataset is crucial for choosing the right partitioning technique. The parameters that we have chosen for describing the dataset distribution are presented below; they represent one of the main contributions of this work and are called *distribution descriptors* in the rest of the article.

Two distinct approaches have been considered: The first one, called *histogram-based*, computes a histogram by superimposing a fixed grid onto the dataset to describe extensively its distribution: Each cell of the grid stores the number of geometries intersecting it. The second one, called *fractal-based*, computes some synthetic parameters deriving from the application of the fractal dimension concept and the Moran's index for capturing in a synthetic way the dataset distribution. Other statistics can be exploited to produce different descriptors; among them we can list the Ripley's K and L functions and other spectral analysis, but we choose the fractal-based ones, since, first, they have already proved to be effective for the partitioning decision problem and, second, we need a representative technique for comparing the feature-extraction based approach with the usage of histograms, that is instead an approach based on row data, thus more deep learning oriented.

Table 2 summarises the symbols used in the formal presentation of the descriptors.

Histogram-based Summarization. Regarding the histogram-based approach, given a spatial dataset D containing geometries, we compute the histogram by choosing a regular grid G and computing for each cell of G the number of geometries of D that it intersects.

Definition 3.1 (Histogram). Given a dataset D, containing a set of geometric features, and a grid $G(n \times n)$ with cell size $r = l/n$ (l being the length of the grid side) and covering the reference space of D (i.e., the MBR of D), the histogram hs^r_D is defined as follows:

$$hs^r_D(i) = count(features\ of\ D\ with\ an\ MBR\ intersecting\ the\ ith\ cell). \quad (1)$$

In the *histogram-based* approach given a dataset D the ordered list of values representing the counts in the histogram cells is used for describing its distribution $(hs^r_D(1), \ldots, hs^r_D(n \times n))$. The

Table 2. Symbols

Symbol	Meaning
D	D represents a spatial dataset containing geometries.
G	G represents the grid used for computing an histogram on D.
n	n is # of cells on one side of the grid G. G has $n \times n$ cells.
l	l is the length of one side of the grid G.
r	r is the width of a cell belonging to G.
$hs_D^r(i)$	it is # of features of D intersecting the ith cell of G with side length r (Definition 3.1).
q	An integer that represents the exponent of the box-counting function.
$BC_D^q(r)$	it represents the computation of the box-counting function with exponent q on dataset D with a grid with cell of width r (Definition 3.2).
α	it is the constant of proportionality used in Equation (3).
E_q	it is the exponent of the power law (see Equation (3)), it represents the fractal dimension of the dataset.
$x_k(i)$	it represents the variable of interest in the computation of the Moran's index.
$\overline{x_k}$	it is the average of the variable of interest computed on all cells of the histogram used in the Moran's index computation.
N	$N = n \times n$ is the total number of cell of the histogram used in the Moran's index computation.
$w_{i,j}$	it represents the weight that is assigned to the pair of cells (i, j) in the computation of the Moran's index. Notice that each cell is identified by a single index i (or j).
EMP_D	it is # of empty cells in the histogram of a spatial dataset D.

histogram is computed efficiently using either Spark (used in this article) or Hadoop as shown in References [9, 30]. The choice of the parameter r can have an impact on the effectiveness of the histograms in representing the dataset distribution. In Section 5.5, we illustrate the results of some specific experiments devoted to the analysis of this issue.

Fractal-based Summarization. Since the list of values in the histogram representation can be quite long, an alternative approach is to use the concept of fractal dimension to describe the dataset distribution by mean of a single number. This approach is usually applied to theoretically infinite set of points and has been extended to finite set of geometries, as proposed in References [6, 8]. Using this idea, given a dataset D a family of histograms are computed and from each histogram a single number is obtained by summing up all the values contained in its cells. This sum is called *box-counting*, and the trend of this function, by varying the size r of the grid cells, provides information about the dataset distribution; in particular, this is straightforward when the dataset presents the self-similarity property (like, any fractal does), which occurs quite often on real datasets. More than one box-counting function can be defined by considering different values for the exponent q, producing different fractal dimensions (E_0, E_2, ...) as theoretically defined in fractal theory.

Definition 3.2. Given a dataset D, containing a set of features, the *box-counting plot* is the plot of $BC_D^q(r)$ versus r in logarithmic scale, where

$$BC_D^q(r) = \sum_i \left(hs_D^r(i)\right)^q \qquad with\ q \neq 1. \tag{2}$$

Now, we can consider such plot and exploit the following observation of Reference [5]: For real datasets the box-counting plot reveals a trend of the box-counting function that, in a large interval

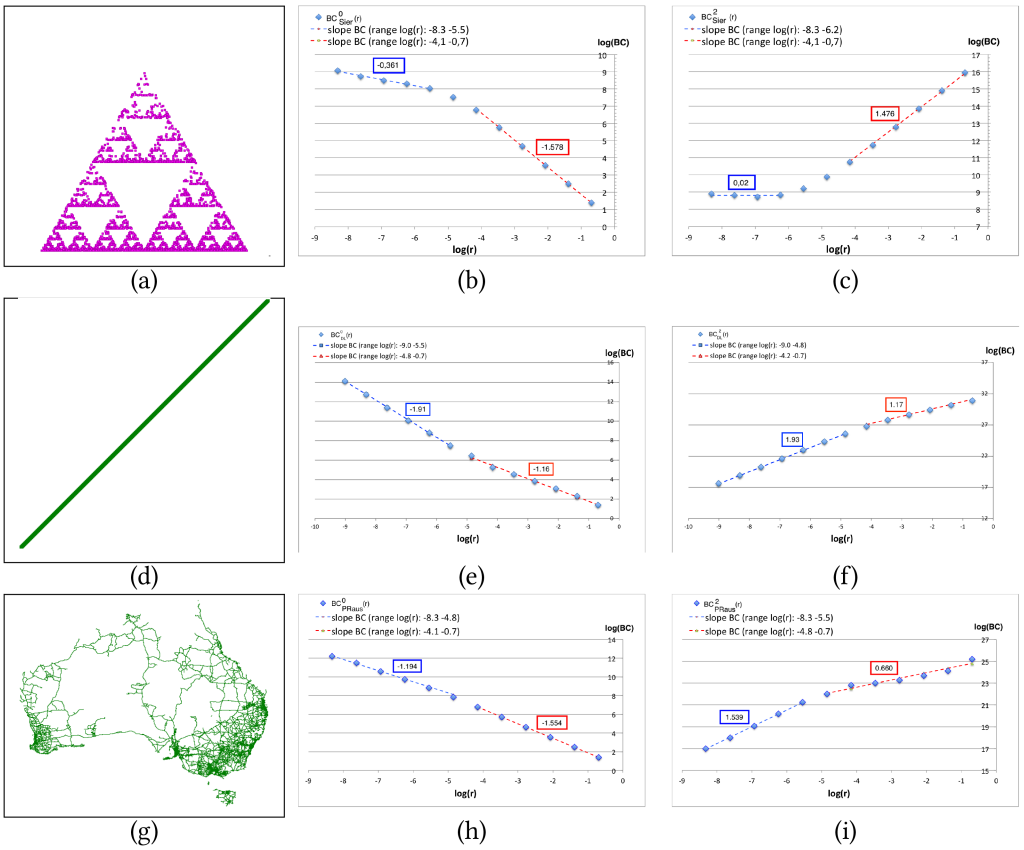


Fig. 5. Example of box-counting plot for ((a)–(c)) a synthetic dataset with the distribution of a Sierpinski's triangle, ((d)–(e)) a synthetic dataset containing a diagonal line with buffer, and ((g)–(i)) a real-world dataset representing the primary roads of Australia.

of scale values r, behaves as a power law:

$$BC_D^q(r) = \alpha \cdot r^{E_q}, \tag{3}$$

where α is a constant of proportionality and E_q is a fixed exponent that characterizes the power law.

The box-counting plot is vital for the computation of the exponent E_q for a given dataset D, since this exponent becomes the slope of the straight line that approximates $BC_D^q(r)$ in a range of scales (r_1, r_2), and thus it can be computed by a linear regression procedure. In our case, we choose to consider the exponents E_0, E_2, and E_3. Figure 5 shows the computation of E_0 and E_2 for some synthetic and real datasets. The first dataset contains small polygons with the distribution of the Sierpinski's triangle, which is a well-known fractal whose dimension is theoretically fixed to the value $log(3)/log(2) \approx 1.585$. The computed value of E_0 and E_2 in this case are very closed to the expected value 1.585. The first part of the plots, for both E_0 and E_2, has a different slope; this is due to the fact that the considered dataset is finite, and thus when the cells of the grid becomes small enough, they will contain only one geometry each, and as a consequence the value of $BC_D^q(r)$ tends to be constant. Also, the second dataset can be described as a fractal with dimension 1, since its distribution follows a straight line representing the diagonal of the reference space.

Also, here the computed slopes are very closed to the expected value. Finally, a real dataset has been considered, representing the primary roads of Australia. In this case, we can notice that the dataset behaves indeed like a fractal, since we can measure slopes in the box-counting plot. Notice that the values of E_0 and E_2 vary according to the considered intervals of values for r (representing the length of the cell side), and for higher values of r they are considerably less than two. This means that the dataset is not uniformly distributed in the reference space.

Again, a MapReduce implementation of this procedure allows the efficient computation of these descriptors as described in References [6, 8].

Moran's Index. Another well-known index that we have adopted for characterizing the dataset distribution is the Moran's index, which is a measure of spatial autocorrelation first presented in Reference [25]. This index is able to detect the grade of autocorrelation regarding a variable of interest x that assumes different values in the cells of a grid, representing the domain of x. In our case, the histograms computed for the previous descriptors E_* are used, and the variable of interest x is represented by the count stored in each cell of the grid in the considered histogram; thus, the reference space where the geometries are embedded represents the domain of x. As shown in the following definition, the Moran's index analyses each cell of the histogram and evaluates how the value stored in the cell is correlated to the values stored in the adjacent cells.

Definition 3.3 (Moran's index). Given a spatial dataset D together with its histogram $(hs^r_D(1), \ldots, hs^r_D(n \times n))$, the Moran's index have been computed considering the variable of interest $x_k = (hs^r_D(i))^k$, with the exponent $k \in \{0, 1\}$. The reasoning behind this choice can be explained as follows: With $k = 0$ the presence (1) or absence (0) of geometries inside a cell is considered; conversely, with $k = 1$ the variation of concentration of geometries inside the cells are evaluated.

$$MI_k = \frac{N}{W} \frac{\sum_i \sum_j w_{i,j}(x_k(i) - \overline{x}_k)(x_k(j) - \overline{x}_k)}{\sum_i (x_k(i) - \overline{x}_k)^2},$$

where:

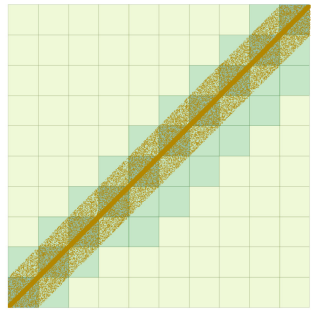
- $w_{i,j}$ is a matrix of spatial weights with zeroes on the diagonal, given a row i it contains ones only for the cells that are adjacent to the ith cell and zeroes everywhere else.
- $N = n \times n$ (i.e., the histogram size)
- $W = \sum_i \sum_j w_{i,j}$ (i.e., the sum of all spatial weights)
- $x_k(i) = (hs^r_D(i))^k$ is the variable of interest in the considered case
- $\overline{x}_k$ is the average of the variable $x_k(i)$.

In general, the typical values of the Moran's index belongs to the range −1,+1. Values near −1 indicates negative spatial autocorrelation (dispersion), while values near +1 means positive spatial autocorrelation (concentration), finally values around zero represent a random arrangement.

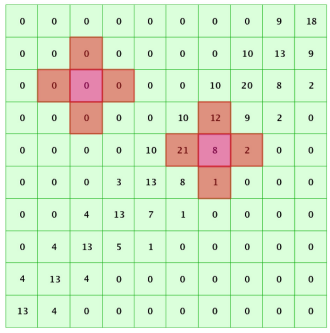
In Figure 6, an example of computation of the Moran's index is shown. We consider the dataset representing a collection of small polygons distributed along the diagonal of the space with a portion of the data that are spread within a given distance (buffer) from the diagonal (an example of this dataset is shown also in Figure 2, second column *Diagonal line*). Notice that the cell on the left provides a positive contribution to the index calculation, since it detects similar values of the variable of interest in its neighbors, while the cell on the right, on the contrary, produces a negative contribution to the index, since very different values of the variable of interest are stored in its neighboring cells.

The MapReduce procedure for computing the descriptors E_0, E_2, and E_3 has been extended to compute also the values of the Moran's indexes: MI_0 and MI_1. In the experiments, to emphasis

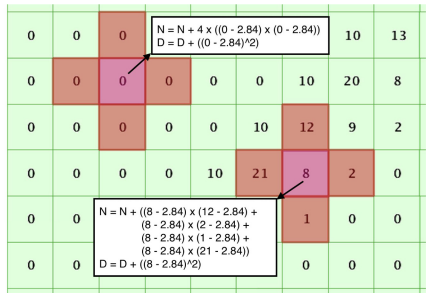
(a) Dataset *Diagonal Line*



(b) One of the computed histograms



(c) Chosen cells for showing the computation of MI_1 (numbers are reduced for sake of readability)



(d) Contributions of the chosen cells to the numerator N and denominator D of MI_1 (average = 2.84)

Fig. 6. Example of Moran's index computation on the Diagonal Line dataset (a). In (b) the considered histogram is shown. In (c) the cells of the histogram are labelled with their count (# geometries they intersect) and two cells are highlighted together with their adjacent cells. Finally, in (d) the contribution of the cells to the computation of the numerator (N) and the denominator (D) of the Moran's index is shown.

the spatial autocorrelation, we introduce also a discretization in five classes of the variable of interest x_1.

Empty Cells. We also consider an additional descriptor that simply counts the percentage of empty cells (cells that are not intersected by any geometry); we call it EMP_D when computed on a spatial dataset D.

In Table 3, the values of M_0, M_1, and EMP for some datasets are shown. Notice that M_0 is often close to 1, since it tends to be influenced by the spatial autocorrelation produced by the fact that empty cells are closed to other empty cells or by the fact that not empty cells are closed to other not empty cells. Thus, a similar value is obtained both for the uniform distribution and the real dataset representing the primary roads of USA. In this situation, M_1 can distinguish the two cases more effectively, but definitively the EMP value separates them clearly. The other two datasets are not well separated by these values, but they are if we consider the other descriptors E_0, E_2, and E_3.

3.3 Evaluation of Quality Metrics

In this section, we briefly describe the quality metrics that characterize the partitioning techniques that we consider in this article, and we show their effect on skewed distributed datasets. We also

Table 3. Computation of the Moran's Indexes M_0, M_1 and Percentage of Empty Cells for the Datasets Presented in Figure 2: (a) a Synthetic Dataset with the Distribution of a Sierpinski's Triangle, (b) a Synthetic Dataset Containing a Diagonal Line with Buffer, (c) a Synthetic Dataset Containing a Double Cluster, and (d) a Real-world Dataset Representing the Primary Roads of USA

Dataset	M_0	M_1	*EMP*
Uniform distribution	0.719	0.011	12.6%
Diagonal line with buffer	0.917	0.739	81.6%
Double cluster	0.847	0.875	87.9%
Primary roads of USA	0.655	0.552	96.7%

describe an efficient way to compute all quality metrics for all partitioning techniques in one Spark job.

Why do we need quality metrics? All big spatial data frameworks that run on multiple machines have to partition the data across machines before being processed. This applies to disk-based systems such as Hadoop, memory-based systems such as Spark, streaming systems such as Storm, key-value stores such as HBase, and big data managements systems such as AsterixDB [17]. Unfortunately, there is no agreement in the community of a single spatial partitioning technique that is universally recommended. The common partitioning techniques are based on grid, R-tree, Quad-tree, and space filling curve. Furthermore, there are many variations under each of these techniques. One of the reasons for having so many spatial partitioning techniques is that the requirements of the systems vary by their architecture and the type of spatial analytics they perform.

To be able to quantify the *goodness* of the different partitioning techniques, several *quality metrics* have been developed. Each quality metric measures one aspect of the spatial partitioning techniques. Depending on the user requirements, one or more of these quality metrics might be chosen to minimize or maximize. The problem is that the quality of the resulting partitions depends on both the dataset distribution and the spatial partitioning technique.

Quality Metrics. To measure the quality of the partitioning techniques when applied to a certain dataset D, we define four quality metrics that have been previously shown to improve the query performance of range query, kNN, and spatial join [11]. These quality metrics are total area (Q_1), total margin (Q_2), total area overlap (Q_3), standard deviation of partition cardinality (Q_4), and average range query cost (ARQ), all defined below.

Definition 3.4 (Total area-Q_1). Given a set of partitions $\mathcal{P} = \{P_i\}$, this quality measure is obtained by computing the sum of the areas of all partitions P_i:

$$Q_1(\mathcal{P}) = \sum_{P_i \in \mathcal{P}} area(P_i.MBR) \cdot P_i.blocks.$$

The multiplication by number of blocks $P_i.blocks$ allows the quality metric to take into account the processing mechanism of big spatial data frameworks. Simply, a partition with multiple blocks is treated by those query processing engines as multiple partitions each with one block. This multiplication ensures that it is counted as multiple partitions.

Definition 3.5 (Total margin-Q_2). Given a set of partitions $\mathcal{P} = \{P_i\}$, this quality measure is obtained by computing the sum of the length of the semiperimeter of all partitions P_i:

$$Q_2(\mathcal{P}) = \sum_{P_i \in \mathcal{P}} semiperimeter(P_i.MBR) \cdot P_i.blocks,$$

where $semiperimeter(MBR) = MBR.width + MBR.height$.

Similarly to Q_1, the multiplication by number of blocks ensures that a partition with multiple blocks is treated as multiple partitions with one block.

Definition 3.6 (Total Overlaps-Q_3). Given a set of partitions $\mathcal{P} = \{P_i\}$, this quality measure is obtained by computing the sum of the area of the overlapping regions produced by intersecting each partition P_i with all other partitions P_j $(i \neq j)$:

$$Q_3(\mathcal{P}) = \sum_{P_i, P_j \in \mathcal{P} \wedge i \neq j} area(P_i.MBR \cap P_j.MBR) \cdot P_i.blocks \cdot P_j.blocks + \sum_{P_i \in \mathcal{P}} area(P_i.MBR) \cdot \frac{P_i.blocks \cdot (P_i.blocks - 1)}{2}.$$

The first term in the equation above calculates the total area of overlap between every pair of different partitions. The multiplication by $P_i.blocks \cdot P_j.blocks$ ensures that each partition is treated as separate partitions each with one block. The second term calculates the overlap that results when we have one partition with more than one block. In this case, if we treat each block as a separate partition, then all blocks will be completely overlapping with all others. Notice that, if P_i has only one block its contribution to the second term is equal to zero.

Definition 3.7 (Standard deviation of partition cardinality (Q_4)). Given a set of partitions $\mathcal{P} = \{P_i\}$, this quality measure is obtained by computing the average of the deviation from the average of the cardinality of each partition:

$$Q_4(\mathcal{P}) = \sqrt{\frac{\sum_{P_i \in \mathcal{P}} (P_i.card - \mathcal{P}.\overline{card})^2}{| \mathcal{P} |}},$$

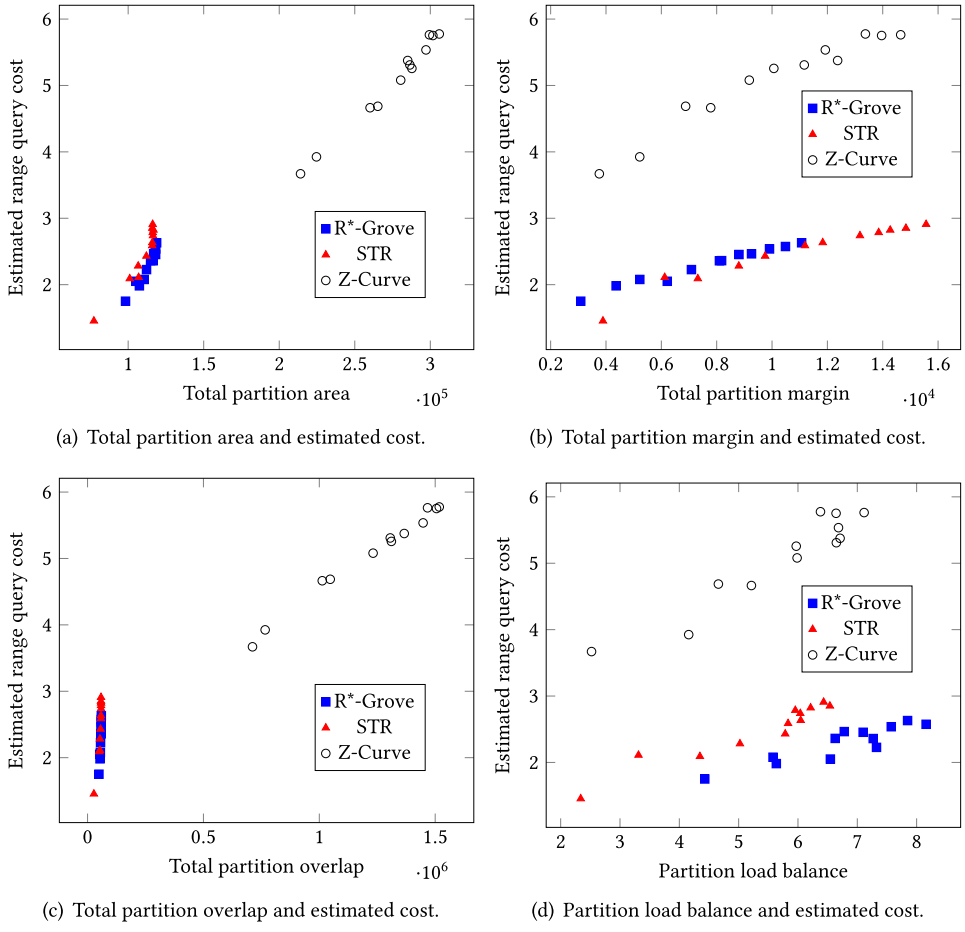
where $\mathcal{P}.\overline{card}$ represents the average cardinality of the blocks of the partitions belonging to $\mathcal{P}$.

Definition 3.8 (Average range query cost (ARQ)). Given a set of partitions $\mathcal{P} = \{P_i\}$, this quality measure is obtained by computing the sum of the average number of blocks that would be scanned if we execute a square query of size $S \times S$ at a random position on the dataset's space that is partitioned by $\mathcal{P}$,

$$ARQ(\mathcal{P}) = \sum_{P_i \in \mathcal{P}} \frac{(P_i.width + S) \cdot (P_i.height + S)}{\mathcal{P}.MBR} \cdot P_i.blocks.$$

Similarly to previous ones, the multiplication by number of blocks is necessary to consider the actual query processing cost as each block is treated as a separate partition.

A previous work [11] proved that the quality metrics Q1–Q4 are good indicators to evaluate the efficiency of a partitioning technique for a specific dataset. In other words, given two partitioning techniques, the one that achieves better quality metrics would provide a better query performance as well. To validate this statement, we carry an experiment that partitions the OSM-Nodes [18] datasets of different sizes by R*-Grove [35], STR, and Z-Curve, as shown in Figure 7. Figure 7(a) and (c) clearly shows that there is a linear relationship between total partition area and overlap with average range query cost. In Figure 7(b) and (d), there are gaps of cost between different techniques, because the cost for a single technique is affected by the combination of all quality metrics. However, there is still an upward trend for each partitioning technique. This observation was also mentioned in Reference [11]. In more detail, Table 4 verifies this observation by showing the correlation values between quality metrics and query performance in different partitioning techniques. These high values validate our claim that we could use partition quality metrics to evaluate performance of a partitioning scheme.



(a) Total partition area and estimated cost.

(b) Total partition margin and estimated cost.

(c) Total partition overlap and estimated cost.

(d) Partition load balance and estimated cost.

Fig. 7. The relation of partition quality and query performance.

Table 4. Correlation between Quality Metrics and Query Performance

Partitioning technique	Total area	Total margin	Total overlap	Load balance
All techniques	0.9853496033	0.4225531469	0.9775651893	0.1635463961
R*-Grove	0.9639500086	0.989214543	0.9639500086	0.9094351766
STR	0.9456511908	0.970685173	0.8974233437	0.9588232937
Z-Curve	0.9927921874	0.9741114139	0.9949534266	0.9516883534

The following part describes how these quality metrics are computed efficiently using Spark.

Quality Metrics Computation. This part describes how we compute a set of quality metrics for a given dataset while considering many partitioning techniques. In our discussion, we borrow some terminology from SpatialHadoop [13] but the approach can generalize to other systems, including Spark-based systems. This step is critical, as it needs to be done for each training dataset that we consider. A naïve approach is to simply partition the dataset using all possible partitioners and then evaluate their quality using all quality metrics. However, this would be too slow and would

limit the performance of the training phase. Rather, we consider a more efficient technique that can accurately calculate the quality metrics without having to actually partition the data. Below, we first describe the notion of a *master file* and then explain how we use it to compute the quality metrics efficiently.

Master files: SpatialHadoop manages the metadata of a partitioned dataset by a small text file, called a master file. The master file of a partitioned dataset contains a list of metadata for all partitions of that dataset. The metadata of a partition includes partition ID, total number of records, partition size, and partition MBR. To execute a query, for example, range query, the query executor would take a look at the master file for early pruning the partitions that certainly do not contribute to the answer. According to previous work, the data partitioning, as encoded in the master file, is the main driving factor for query performance [13].

The key observation is that we can produce many master files for all partitioning techniques in one Spark/MapReduce job without having to actually partition the data. In particular, the quality metrics that we consider in this article are total area ($Q1$), total margin ($Q2$), total overlaps ($Q3$), or average range query cost (ARQ) of all partitions, as we mentioned in Section 3.3. All of those metrics could be computed from the master file of the partitioned dataset, as they only require the MBR and total size of each partition. Furthermore, other researchers can easily extend these quality metrics based on the demands of the desired analytic operation, e.g., standard deviation of partition size, and disk utilization.

Efficient Computation of Master Files: To create a training data point, we have to find the best partitioning technique among several options (e.g., kd-tree, R*-Tree, STR, Grid, and Z-Curve) in terms of a specific quality metric, for example, total area of all partitions. Instead of physically partitioning the data using these techniques, we observe that all information encoded in the master files, i.e., MBR and total size, are associative and commutative aggregate functions that can be computed in a local/global manner without the need to group all records of one partition in one machine. In other words, instead of partitioning the data into partitions and then computing those aggregate functions, we can directly compute these aggregate functions. Simply, each machine computes local values for *all partitions* and then they are grouped by partition ID to be further aggregated into final values. Furthermore, we can compute these aggregate values for *all* partitioning techniques in one job by extending the grouping key to be ⟨partition*er* ID, partition ID⟩.

Once all the master files are computed, we can then compute the quality metrics (i.e., $Q1$, $Q2$, $Q3$, and ARQ) as described above on a local machine since the size of the master files is sufficiently small.

3.4 Model Training

In this section, we describe how we conduct a deep learning model that is able to predict the best partitioning technique for a spatial dataset in terms of a specific quality metric. The first challenge that we have to address is to choose a suitable deep learning algorithm for this problem. As we mentioned, the partitioning selector problem is analogous to image classification problem. Thus, we could consider several novel classification models such as Convolutional Neural Network (CNN) or a fully connected neural network (FC). If the number of data points is large enough, e.g., millions of data points, then CNN would mostly outperform a fully connected model. However, this might not be applicable for our system, where the number of data points is only in the thousands. Based on a previous work [27], we carried an experiment for algorithm selection at Section 5.2. Finally, we chose a fully connected neural network to train and test our model. Figure 8 shows the architecture of a fully connected neural network that we use for spatial partitioning selection

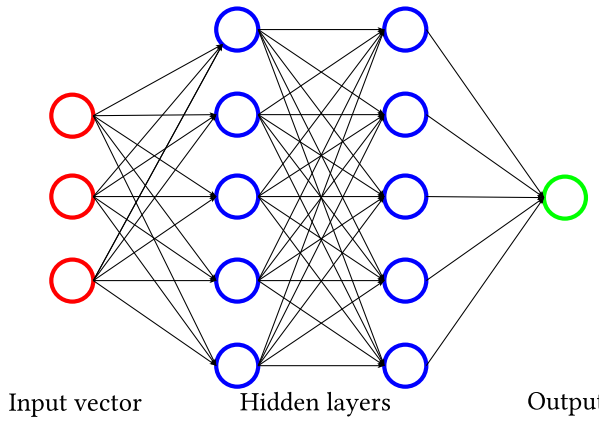


Fig. 8. A sample fully connected model that we adopt in the article. In this work, we vary the number of hidden (blue) layers and the number of hidden units per layer.

model. The input vector (X) consists of a summary of the input dataset, either the histogram or the fractal-based descriptors. In particular, the vector X is composed of the following.

Histogram-based vector: In this case, the vector contains exactly all the counts collected in the cells of the histogram computed on dataset D:

$$X = \left\langle hs_D^r(1), \ldots hs_D^r(n) \right\rangle.$$

This means that the size of the input layer is always equal to the size of the histogram, i.e., number of bins in the histogram.

Fractal-based vector: In this case, the vector contains the descriptors computed on dataset D:

$$X = \langle \mid D \mid, E_0(D), E_0'(D), E_2(D), E_2'(D), E_3(D), E_3'(D),$$

$$M_{0_{min}}(D), M_{0_{avg}}(D), M_{0_{max}}(D), M_{2_{min}}(D), M_{2_{avg}}(D), M_{2_{max}}(D),$$

$$M_{3_{min}}(D), M_{3_{avg}}(D), M_{3_{max}}(D), EMP_D \rangle,$$

where (i) $E_0(D)$ is the exponent E_0 computed for a dataset D, and we have two values, E_q and E_q', for each exponent, since in many cases the behaviour of the dataset follows trends similar to those shown in Figure 5; (ii) $M_q(D)$ is the Moran's index computed for the variable of interest $(hs_D^r(i))^q$, and here we use three values, since we consider a family of histograms and thus we produce several values for $M_q(D)$. Thus, we take the minimum, maximum and average value for representing the behaviour of the dataset D.

The hidden layers are fully connected, and we vary their sizes in the experiments section to tune the system. The size and number of hidden layers can be tuned differently according to which summarization technique we use. The output vector is a single categorical value, which is the best partitioning technique among Kd-tree, R*-Grove, STR, Z-Curve, Grid, and RR*-Tree. The output value is encoded as a number in range $[0 - 5]$, which is the order of the corresponding partitioning technique. We choose to build a separate model for each quality metric as each one of them might need to catch different aspects of the input vector.

The activation function for hidden units is *ReLU* function, except for the last layer, where we use $softmax$ function. Since the output value is categorical, we use $categorical\_crossentropy$ as the loss function.

As recommended in deep learning, we separate the input dataset into three parts: *training*, *validation*, and *testing*. The training set is used to train the model and adjust the weights on all the connections in the neural network. The validation set is used during the training phase to evaluate the current model and avoid overfitting. The validation set is *never* fed through the input layer, so it is always a *new* dataset to the model. Finally, the test dataset is used for final evaluation as shown in the experiments section.

To give the network enough time to stabilize without overfitting, we periodically measure the accuracy of both the training and validation sets. When the accuracy of the validation set stops improving or starts to drop, it is a signal of overfitting. Therefore, we terminate the training phase and retrieve the last good model right before the accuracy dropped.

4 APPLICATION PHASE

In this phase, the system takes a dataset D that was not inspected earlier by the framework and a quality metric (QM). The goal is to predict which partition technique (PT) among the ones considered by the framework will produce the best behaviour in the chosen quality metric QM. The main challenge of this step is that is has to be much faster than applying all partitioning techniques and choosing the best. This phase works in two steps.

The first step summarizes the data to produce a fixed-size vector (X') that describes the input data distribution as described in Section 3.2.

The second step feeds the vector (X') computed in the first step into the machine learning model (M) that corresponds to the quality metric QM. The output of the model is a label (Y') that simply names one of the partitioning techniques (PT) that is estimated by the model to produce the best quality metric (QM). The selected partitioning technique is then passed to any big spatial data system, e.g., SpatialHadoop or GeoSpark, to actually partition the data. In other words, our framework does not actually partition the data, it just chooses a partitioning technique to apply and it is up to the user to choose how to apply it.

More specifically, in the following subsection the application of the system to the spatial join operation is illustrated by considering a test case with real datasets.

4.1 Application of the Model in a Real System

To show how the proposed model can be applied in a real system, we show in Figure 9 the flow chart describing the necessary steps to perform a given operations OP, for example, a spatial join, on a pair of datasets, D_1 and D_2, with unknown distributions. In the figure, the optimization task is composed of the following steps:

(1) **Histogram computation:** For each dataset D_i, the corresponding histogram H_i is computed, representing the input vector X'; the cost of this operation is denoted as $COST_H(D_i)$.
(2) **Quality metric choice:** Given the operation to execute OP, the corresponding quality metric QM_x is chosen; the cost of this operation is trivially close to zero.
(3) **Partitioning technique choice:** Given QM_x, the corresponding model NN_x is activated passing as input the vector X', obtaining the suggested partitioning technique PT_i, one for each input dataset. The cost of this operation is again close to zero, thanks to the trained machine learning model NN_x.
(4) **Partitioning:** Each chosen technique PT_i is applied to the corresponding dataset D_i, producing a partitioned dataset PD_i.
(5) **Operation computation:** The operation OP is executed on the partitioned datasets PD_i; the cost of this execution is denoted as $COST_{OP}(PD_1, PD_2)$.

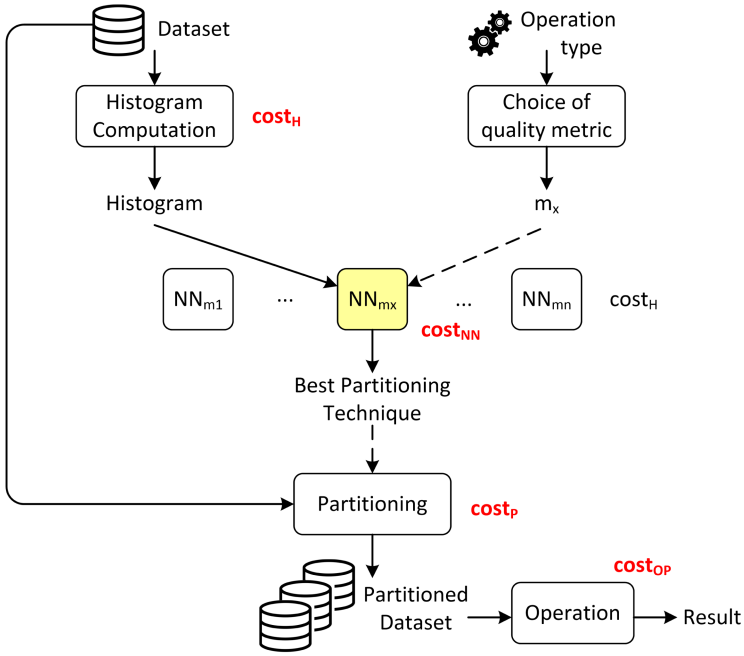


Fig. 9. Flow chart of the optimization task.

The application of the proposed approach is convenient, since the following conditions are very often satisfied in particular for the spatial join operation ($\triangleright\!\triangleleft$); in this case the quality metric is total margin (QM_{TM}):

- **Basic condition:** The cost for generating the histogram (i.e., the input vector X') for a given dataset must be significantly less than the cost of partitioning the same dataset:

$$COST_H(D_i) << COST_P(D_i).$$

  This operation is preformed by a parallel task implemented in Spark and as shown in Figure 18 of the next section, this cost is an order of magnitude less than the cost for partitioning a dataset.
- **Specific condition for $\triangleright\!\triangleleft$:** the average cost of the execution of OP on the partitioned datasets must be less than the cost of executing it on the original datasets:

$$COST_{\triangleright\!\triangleleft}(PD_1, PD_2) < COST_{\triangleright\!\triangleleft}(D_1, D_2).$$

- **Optimization condition for $\triangleright\!\triangleleft$:** The average cost of the optimization phase must be less than the gain produced by the optimization:

$$COST_H(D_1) + COST_H(D_2) + COST_P(D_1) + COST_P(D_2) < COST_{\triangleright\!\triangleleft}(D_1, D_2) - COST_{\triangleright\!\triangleleft}(PD_1, PD_2).$$

To test the application phase and verify the satisfaction of the second and the third conditions, we performed some experiments in a specific case using real datasets. In particular, we consider two real datasets, D_{PRoads} and D_{Builds}, containing the primary roads and buildings of the USA, respectively. Each dataset has been partitioned by applying the six considered techniques. Then, the spatial join between all possible combinations of partitioned datasets has been performed (in total 36 joins). In Table 5, the execution time in seconds of each combination is shown.

Table 5. Execution Times of the Spatial Join Operations in Seconds

Combinations of part. tech.	D_{Builds}					
D_{PRoads}	**Grid**	**Kd-Tree**	**RR*-Tree**	**R*-Grove**	**STR**	**Z-Curve**
Grid	413.2	303.4	245.0	259.6	293.1	279.2
Kd-Tree	318.2	172.3	142.8	148.3	134.9	217.0
RR*-Tree	325.2	208.3	159.4	160.0	148.6	232.9
R*-Grove	357.8	212.3	170.7	171.9	166.9	217.0
STR	343.8	200.2	312.2	16.95	145.2	292.7
Z-Curve	361.0	251.5	220.2	217.2	229.2	235.3

All possible combinations of partitioning techniques applied to datasets D_{PRoads} and D_{Builds} are considered.

Finally, considering the pair (RR*-Tree, RR*-Tree) chosen by the proposed machine learning model NN_{TM} (i.e., the neural network for the chosen quality metric: total margin), we can observe that (i) NN_{TM} is able to detect the pair that is in the top positions in the ranking of spatial join time execution; (ii) the gain with respect to the join performed on the original datasets is about 99.4%; (iii) the gain with respect to the average performance of all pairs is about 31.9%; and, finally, the gain with respect to the worst pair is about 61.4%. The gain obtained by applying the suggested partitioning techniques is as follows:

$$COST_{\triangleright\triangleleft}(D_1, D_2) - COST_{\triangleright\triangleleft}(PD_1, PD_2) = 28,859\,\text{s}$$

and the cost of optimization is

$$COST_H(D_1) + COST_H(D_2) + COST_P(D_1) + COST_P(D_2) = 1,238\,\text{s}.$$

This results allow us to confirm that the above-mentioned conditions are all satisfied. A wider analysis of applicability considering other operations is out of the scope of this article, also because previous works [11] about spatial partitioning techniques already confirm the effectiveness of their use.

A tutorial showing the steps for applying the proposed system is available at: https://github.com/tinvukhac/deep-spatial-partitioning.

5 EXPERIMENTS

This section provides the details of our extensive experimental evaluation. The goal of this experimental evaluation is to measure how accurate the proposed approach is in choosing the best partitioning technique. The experiments will also compare the two summarization techniques to verify which one is more effective for this problem. In the rest of this section, Section 5.1 provides the experimental setup. Section 5.3 describes how we tune the deep learning model. Then, we evaluate the accuracy of the proposed model for both synthetic and real data in Section 5.4. Section 5.5 illustrates the effect of histogram size to the model accuracy, model complexity and training cost. After that, Section 5.6 shows the effect of the dataset size on quality metrics and justify the medium sizes of the synthetic datasets that were used for training. Section 5.7 will focus on evaluating the performance of the system in terms of running time considering both the creation of the training set on one side and the computation of the Histograms-based and Fractal-based summarizations on the other side. Notice that, in the fractal-based summarization we also include the Moran's index. Section 5.8 considers the effect of including in the training set also collection of data with oblong rectangles.

Table 6. Experiments and Performance Metrics

Experiment	Parameters	Metrics
Model tuning	# of hidden layers, units	Accuracy
Model accuracy	dataset distribution	Accuracy
Histogram effect	histogram size	Accuracy, model complexity
Stability of quality metrics	dataset size, HDFS block size	Quality metrics
Summarization performance	dataset size	Running time

5.1 Experimental Setup

Table 6 shows the list of experiments that we are carrying out. (1) First, to tune the parameters of the deep learning models, we vary the number of hidden layers and the number of units per layer to find a suitable fully connected architecture for each summarization technique and for each quality metric. (2) Second, we conduct several experiments to see how the model learns to predict the best partitioning technique from training data for both synthetic and real data. (3) Third, we vary histogram size to see how it affects the model accuracy and complexity. In particular, for each histogram size, we feed the data to several models with different number of hidden layers/units to see which configuration is suitable for a specific histogram size. This experiment explains how we choose histogram size and model architecture for the second experiment. (4) Fourth, to justify the parameters that we use for synthetic data generation, we show the stability of the quality metrics as the dataset size varies. This allows us to generate many medium-size synthetic datasets to save time instead of generating a few large datasets. (5) Fifth, we measure the running time of the summarization process to show that the proposed solution can be applied in practice, since the required effort is significantly less than the cost for partitioning the dataset with all six available techniques. (6) Sixth, we measure the effects on model accuracy of the addition to the training set of new synthetic datasets containing oblong rectangles.

We run our experiments on a cluster of one head node and 12 worker nodes, each having 12 cores, 64 GB of RAM, and a 10 TB HDD. They run CentOS 7 and Oracle Java 1.8.0_131. The cluster is equipped with Apache Spark 2.3.0 and Apache Hadoop 2.9.0. We implement our deep learning model on Keras [19] with TensorFlow 1.12.0 as the backend.

Datasets: In Table 7 the characteristics of the generated synthetic datasets are presented [36]. Notice that for training the model, the generated datasets do not have to be very big. They just have to be diverse enough to represent various characteristics of real data. In Section 5.6 below, we justify this decision by showing the independence of the relative quality of partitioning techniques with dataset size. For each distribution, we generate 100 different datasets with different seeds. The collection contains 1,600 datasets with about 210 millions of geometries in total.

Table 8 shows the real datasets that we use for testing the model. All datasets are publicly available through the SpatialHadoop website [13]. We picked three datasets, buildings, lakes, and roads. To have a decent number of datasets with different distributions, we split each dataset into five parts that roughly enclose North America, South America, Europe, Africa, and Asia+Australia. The size of each part is shown in the table.

Notice that the aim of this experiment session is to verify the quality of the model, i.e., to test the performance of the neural network that predicts the right technique to choose to obtain the best partition with respect to a given quality measure. We are not testing the impact of the choice on the final operation that is applied by the user on the partitioned datasets.

Training sets: We produced the data points of the training sets from the generated synthetic datasets, listed in Table 7. For deep learning, a data point is a pair (X, Y), where X represents the

Table 7. Training Set Generation: Datasets of Different Distributions Generated for the Training Phase

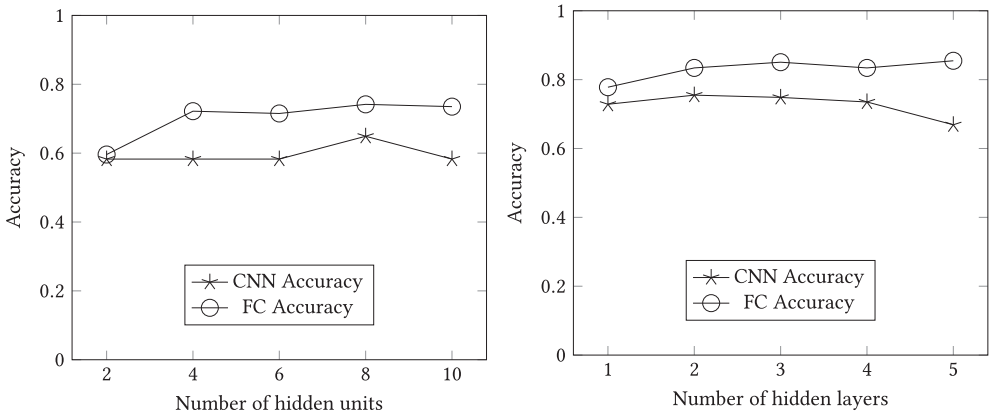
Distribution	Num. of Datasets	Size	Num. of Features
Uniform	100	512 Mb	7,000,000
Linear	100	512 Mb	7,000,000
Linear rotated	100	420 Mb	6,000,000
Diagonal	100	1.1 Gb	15,000,000
Diagonal rotated	100	1.1 Gb	15,000,000
Parcel	100	512 Mb	7,000,000
Cluster	100	1.0 Gb	5,000,000
Linear/Linear rot.	100	1.0 Gb	11,000,000
Linear/Uniform	100	1.0 Gb	14,000,000
Linear rot./Uniform	100	1.0 Gb	11,000,000
Diagonal/Diagonal rot.	100	2.2 Gb	30,000,000
Diagonal/Uniform	100	1.6 Gb	22,000,000
Diagonal rot./Uniform	100	1.6 Gb	22,000,000
Parcel/Uniform	100	1.0 Gb	14,000,000
Parcel/Linear rot.	100	1.0 Gb	13,000,000
Cluster/Linear rot.	100	1.5 Gb	11,000,000

Table 8. Real Datasets Used for Testing

Dataset	Num. of Features
Buildings-1	1,393,451
Buildings-2	8,708,373
Buildings-3	91,657,814
Buildings-4	7,925,531
Buildings-5	5,111,326
Lakes-1	828,221
Lakes-2	4,246,874
Lakes-3	2,072,660
Lakes-4	619,689
Lakes-5	652,583
Roads-1	3,672,499
Roads-2	19,729,459
Roads-3	33,078,006
Roads-4	6,725,578
Roads-5	9,137,471

summarization of one dataset (using one of the two proposed summarization techniques) and Y represents the corresponding best partition. Since we have in total five quality metrics and two summarization techniques, histograms-based and fractal-based, for each dataset we produce 10 data points, one for each training set dedicated to one model: $(X_H, Y_i)_{Q_i}$, $1 \leq i \leq 5$ for the histograms-based summarization and $(X_F, Y_i)_{Q_i}$, $1 \leq i \leq 5$ for the fractal-based one.

In total, in each training set we have 1,600 data points generated from the synthetic datasets. Unless otherwise mentioned, we use 80% of generated data points as training data and the other



(a) Accuracy and # of hidden units on CNN and FC model (b) Accuracy and # of hidden layers on CNN and FC model

Fig. 10. Algorithm comparison between the CNN and FC models.

20% as the testing data. Of the 80% training set, 20% of it is used as a validation set, i.e., 16% of the overall data.

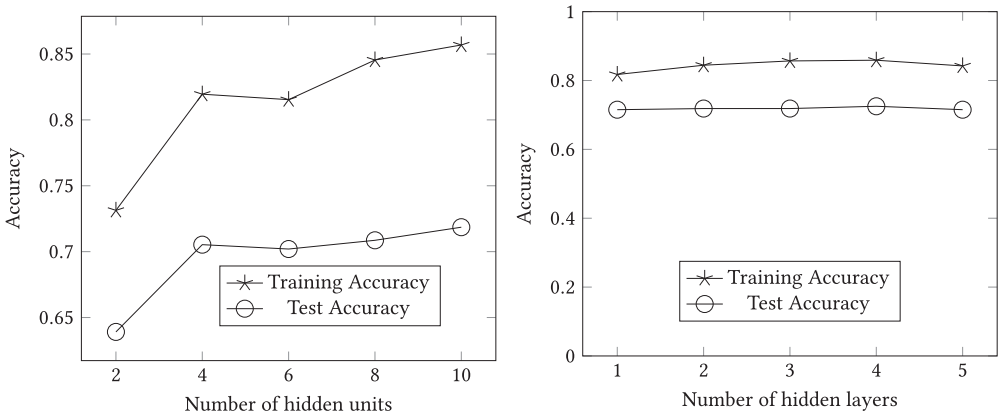
Accuracy Metrics: We use a Boolean accuracy metric. That is, we compare the label generated by the model with the true label that was selected by computing the actual quality metrics and choosing the best. If they match, then the accuracy is 1.0; otherwise, it is 0.0. Then, we take the average over all the test set. Notice that since we have six possible labels that correspond to the six partitioning techniques, a completely random baseline would have an accuracy of $1/6 \approx 17\%$.

5.2 Algorithm Selection

In this experiment, we compare two approaches in the context of spatial partitioning selection problem: the CNN model and an FC model. The choice of these two candidate models is inspired by the observation that our problem is analogous to an image classification problem. We are trying to set up the same parameters, e.g., number of hidden layers and number of hidden units in each layer for both models. After that, we train these models and evaluate their accuracy. Figure 10 shows that FC models outperform CNN models in terms of model's accuracy. Furthermore, CNN also requires more training time before its convergence point for a same given dataset. The reason is that CNN typically requires a training and testing set with very large number of data points. However, the training set we created from histograms is limited in size, which might be more suitable to a simple architecture like FC models. Finally, we chose to use FC model for our following experiments. In our published repository, we provide the implementation for both the CNN and FC models. Therefore, users could choose any model that is suitable for their own datasets.

5.3 Model Selection

This experiment shows our effort to find the suitable model architecture for our training datasets. As we mentioned in Section 3.4, there are two options to generate training dataset with data points (X, Y). First, X could be the flatten vector of the histogram matrix, which is chosen as 50×50 in this experiment (see Section 5.5 for more details about this choice). Second, X can be considered as the ordered skewness values that are computed by fractal-based summarization methods (fractal dimensions and Moran's indexes). Y is the single number that reflects the order (base 0) of the best partitioning options among Kd-tree, R*-Grove, STR, Z-Curve, Grid, and RR*-tree.



(a) Accuracy and # of hidden units with histogram input (b) Accuracy and # of hidden layers with histogram input

Fig. 11. Tuning model parameters for the *histogram-based* summarization technique.

In this experiment, we use total partition area as reference quality metric to evaluate partitioning techniques. The best technique should have the smallest total area. The different kinds of feature vector might require different configurations of the learning model. Moreover, we use a fully connected neural network and vary both the number of hidden layers and the number of units per layer to find the suitable model for each kind of input vector.

Figure 11(a) shows the accuracy of a fully connected model with three hidden layers for the training dataset with histogram vector as the input vector. We vary the total number of units in each layer to see how the accuracy changes. When the number of hidden units is small, e.g., 2, the model is not able to capture the complex information from training data. As the number of hidden units increases, the accuracy for both training and testing process are stabilized. Thus, we choose 10 as the number of hidden units for each layer.

In the next experiment shown in Figure 11(b), we fix the number of units per layer as 10 and then vary the number of hidden layers to see how it affects the model accuracy. We observe that the accuracy is stable when the number of layers changes with the best value at 3 hidden layers. Based on these two experiments, for the model with an input vector composed of the flatten representation of the histogram matrix with size 50×50, we choose the fully connected model with 3 hidden layers and 10 hidden units per layer.

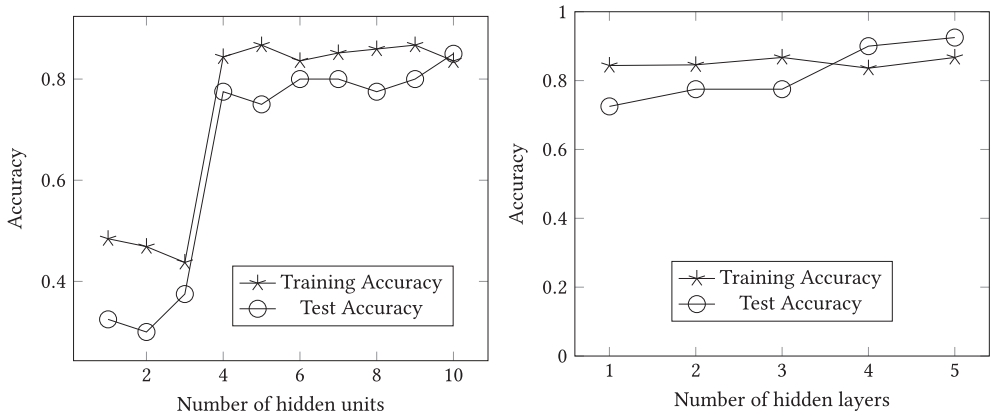
We repeat the same procedure with the other summarization technique, i.e., the fractal-based one, as shown in Figures 12(a) and (b). In this case, we choose an architecture with three hidden layer and five hidden units in each layer.

In these experiments, we can also observe that the model can reach up to 90% and 80% accuracy when applied on synthetic and real test data, respectively, which shows the applicability of the proposed approach to the problem of spatial partitioning.

5.4 Model Accuracy

This section shows the accuracy of our model to predict the best partitioning technique for datasets with different distributions including synthetic and real datasets. We only use the synthetic datasets for training, and we use both synthetic and real data for testing, reporting their accuracy separately.

In such experiments, we measure the accuracy of our predictive models considering two configurations: (i) In the first one, the input vector is the ordered list of skewness values, which are



(a) Accuracy and # of hidden units with skewness input (b) Accuracy and # of hidden layers with skewness input

Fig. 12. Tuning model parameters for the *fractal-based* summarization technique.

computed by fractal-based summarization methods, while (ii) in the second one the flatten vector representing the histogram of the dataset is the input.

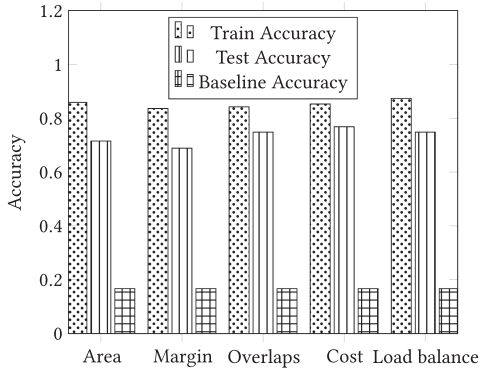
The quality metrics include the following: total area (Q_1), total margin (Q_2), total overlaps (Q_3), the standard deviation of the partition size (Q_4), and the average range query cost (ARQ) of partitioned datasets. We evaluate such metrics in six different partitioning techniques: Kd-tree, R*-Grove, STR, Z-Curve, Grid, and RR*-tree. Generally, if we randomly choose a technique between those options, then the probability that we can choose the best one is 17%. Since there are no similar work that exists in literature, we choose this number as the baseline accuracy to compare with our proposed method, which is a common practice in machine learning evaluation.

Figure 13(a) and (b) shows the accuracy of training and testing process when we train and test our model with data points coming from synthetic datasets. As we can observe, in both configurations, the models can predict the best partitioning technique for different quality metrics with an accuracy of up to 78%, which is significantly better than the baseline method.

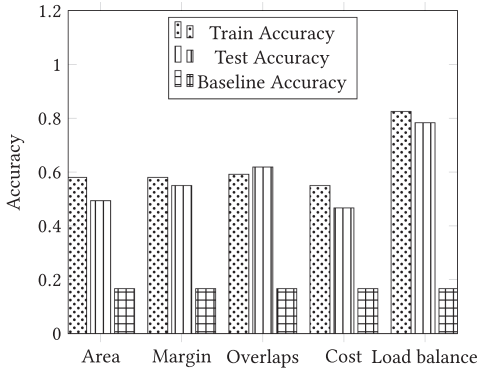
Figure 13(c) and (d) shows the accuracy of training and testing process when we train our model on data points coming from synthetic datasets, and *test it on data points from real datasets*. Although the test accuracy is not as high as the synthetic datasets, it still gives us a good accuracy with up to 64%. Keep in mind that the model was trained on synthetic data only and the real datasets used in testing are observed by the model for the first time.

Comparison of the two summarization techniques: One interesting observation in this experiment is that the histogram-based summarization outperforms the fractal-based skewness measures developed by the experts for *synthetic data*. However, when it comes to real data, the results for both summarizations are very similar (only in some cases the experts' measures outperform the simple histogram).

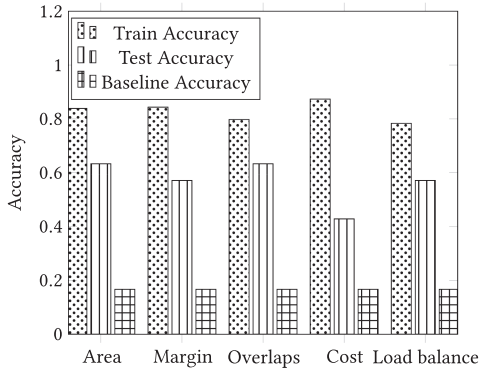
This indicates that the deep learning model can learn and produce an accurate model for the datasets it sees during the training phase and can outperform existing methods. However, the skewness measures, as developed by experts, are good at extracting meaningful measures of skewness and allow to find hints of similarity between two datasets that, from a simple comparison of their plots, might seem very different. In the histogram configuration the models learn to detect similarity between datasets mainly considering a visualization-based comparison working at the granularity of the histogram. This implies that the model that learns from histograms need a train-



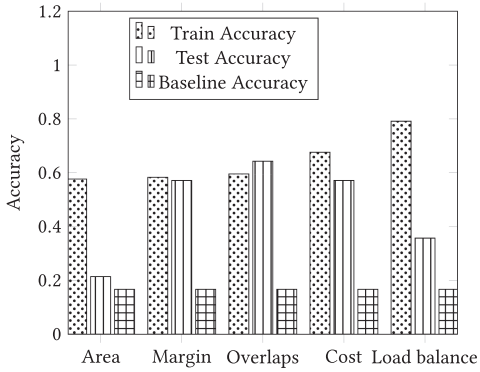
(a) Histogram summarization with test on **synthetic** data



(b) Skewness summarization with test on **synthetic** data



(c) Histogram summarization with test on **real** data



(d) Skewness summarization with test on **real** data

Fig. 13. Model accuracy when train and test on synthetic and real datasets.

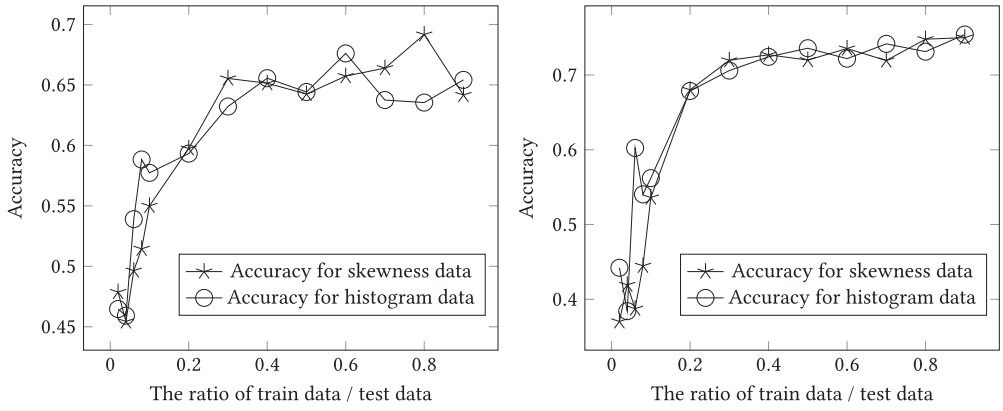
ing set containing a higher variety of distributions and datasets with a higher similarity to the real ones to increase its accuracy.

We expect that if we have a bigger training set with more diverse synthetic datasets, then the deep learning approach with histogram can produce better results. We plan to verify this conjecture in future works by adding more distributions and more datasets to the training set.

Figure 14 shows the accuracy of the predictive models with skewness and histogram input when we vary the ratio between number of train and test data points. As expected, the accuracy increases and then stabilizes as the ratio of the training set increases. This verifies that the predictive models are able to capture the characteristics of the input datasets and that they get more accurate with more training points.

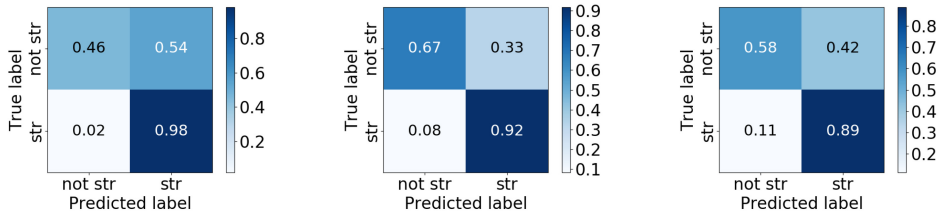
Another accuracy metric that is usually used in multi-labeled deep learning models is the *confusion matrix*. As shown in Figure 15, this matrix shows for each pair $(label, metric)$, a square divided in four parts containing the percentage of (i) true positive cases obtained in the test (lower right sub-square), (ii) true negative cases (upper left sub-square), (iii) false positive (lower left sub-square), and (iv) false negative (upper right sub-square).

Figure 15(a)–(e) shows the five confusion matrices regarding the *STR* partitioning technique, one for each each quality metric. Notice that for this technique, which is the one having more samples in the training set, the true positive percentage is always over 89%. However, for the other

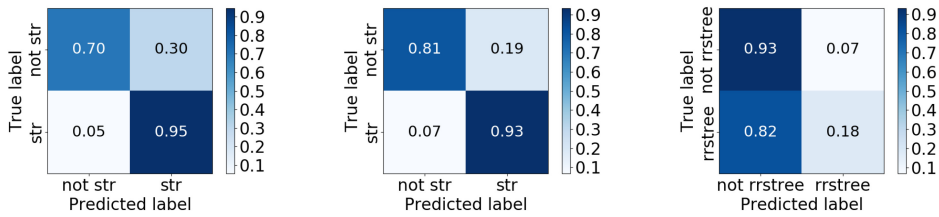


(a) Predict the technique with best total area (b) Predict the technique with best total margin

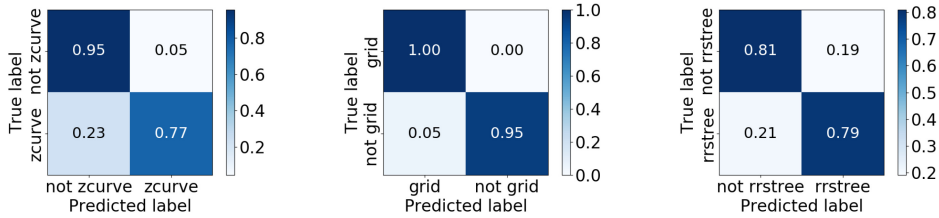
Fig. 14. Model accuracy when varying the ratio of train data/test data.



(a) STR - Q1 Total Area (b) STR - Q2 Total Margin (c) STR - Q3 Total Overlap



(d) STR - Q4 Load Balance (e) STR - Q5 ARQ Cost (f) R*-tree - Q1 Total Area



(g) ZCurve - Q2 Total Margin (h) Grid - Q4 Load Balance (i) R*-tree - Q5 ARQ Cost

Fig. 15. Confusion matrices for different index techniques and considering different quality measures.

techniques we do not reach the same optimal results. For instance, considering the *RR*-tree* no test cases are available for such technique with the load balance metric, and in the other metrics the results are varying: good for the range query cost metric (Figure 15(j)), but not as good for the total area metric (Figure 15(f)). The same is true for other techniques, for instance, the *Grid* technique show very good results with the load balance metric (Figure 15(i)), but for the others no test cases are available.

The main reason for the above-described results is that many techniques (but *STR*) are underrepresented in the training data, which is a known problem that causes machine learning models to be incapable of learning their characteristics. In our problem, it was not easy to balance the training data, as we cannot directly specify which partitioning technique is the best for a specific dataset; rather, we generate different synthetic data and run them through all the partitioning techniques, and the best is selected based on their behavior.

5.5 The Effect of Histogram Size

In this section, we study the effect of the histogram size. Since the histogram size controls the size of the input, the optimal model parameters, i.e., number and size of hidden layers. Therefore, for each histogram, we repeat the model tuning experiments described in Section 5.3, and we report here the results of the optimal model.

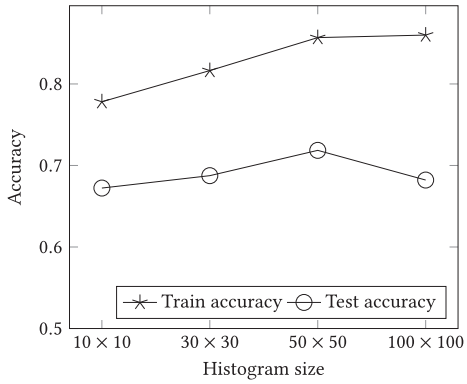
Figure 16 reports the accuracy of the best model found as the histogram sizes from 10×10 to 100×100. This experiments shows the tradeoff between the model complexity and accuracy. One one hand, when the histogram size is small, the model also tends to be small but can be trained accurately. On the other hand, when the histogram is large, the model becomes more complex, but it cannot be trained accurately given the amount of training data that we have. The histogram size of 50×50 tends to strike a balance between these two.

Figure 16(b) further confirms this observation by showing the optimal model parameters that we found for each histogram size, i.e., number of hidden layers and size of the hidden layers. For the largest histogram size, the neural network model becomes more complex with more layers and more neurons per layer. We expect that if there are more training data, then a larger histogram size could be more suitable. Additionally, we also show in Figure 16(c) that a 100×100 histogram requires a significantly 2.5 times longer to stabilize as compared to the 50×50 histogram, which is also attributed to the complexity of the model.

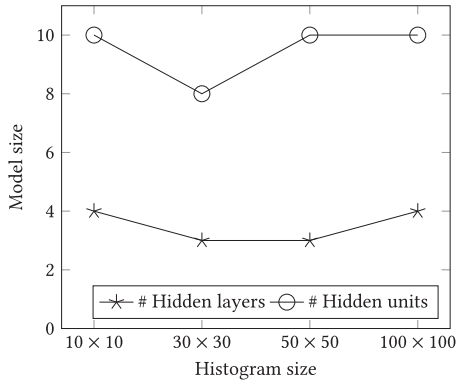
5.6 Stability of Quality Metrics

In this experimental evaluation, we used moderate-size synthetic data with about 1.0 GB each. Although the real datasets can be arbitrarily large, we chose to keep the synthetic datasets small to be able to generate many datasets in a short time. We experimentally show in this part that this is still a valid approach by showing that the relative performance of the synthetic datasets is the same regardless of the size. Which means that the deep learning model will see no difference between the small and big datasets in terms of which partitioning technique is better as long as the distribution is fixed.

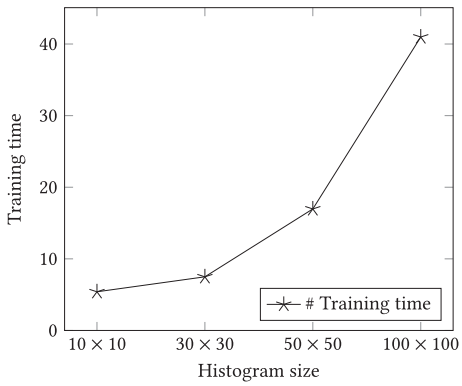
In the experiment shown in Figure 17, we fix the distribution type to the diagonal dataset, and we vary the generated dataset size from 5 to 80 GB. As we increase the dataset size, we also increase the block size to ensure that the number of blocks is roughly the same for a fair comparison. For example, when we increase the size from 5 to 10 GB, we also increase the block size from 16 to 32 MB. We evaluate the performance of four partitioning techniques (Kd-tree, R*-Grove, STR, and Z-Curve) in two quality metrics (Q1, total area, and Q2, total margin). The main observation from Figure 17 is that the quality measures of partitioning technique do not change as long as the ratio of dataset size/block size remains constant. Therefore, the best partitioning technique (STR) is



(a) Accuracy

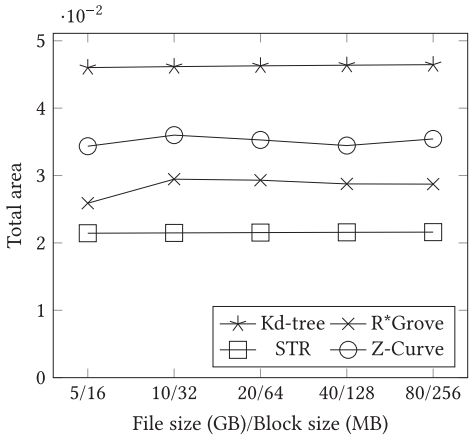


(b) Best model parameters

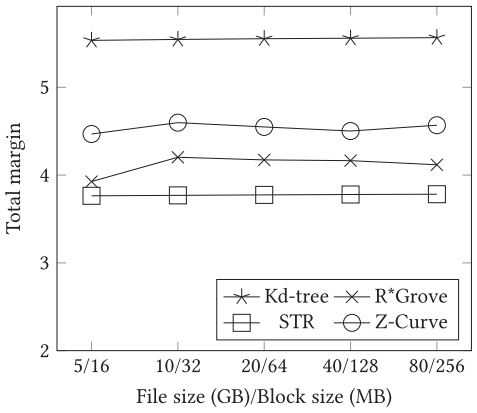


(c) Training time in seconds

Fig. 16. The effect of histogram size.



(a) Total area



(b) Total margin

Fig. 17. Stability of quality metrics.

Table 9. The Independence of the Best Index in Terms of Total Area and Dataset Size

Dataset size(GB)	Kd-tree	R*-Grove	STR	Z-Curve	Best Index
20	0.057	0.030	0.026	0.032	STR
50	0.039	0.028	0.029	0.034	R*-Grove
100	0.037	0.030	0.020	0.041	STR
200	0.036	0.032	0.032	0.050	STR

also consistent over different dataset sizes as well. Given this result, instead of spending hours to compute the best partitioning technique of a dataset with size 128 GB with normal HDFS block size (128 MB), we could execute the same operation for a dataset size of 1 GB with HDFS block size 1 MB and get the same result. This observation allow us to significantly reduce the time to generate our training data points. In practice, we generate the training data points from datasets in Table 7 with HDFS block size 4 MB.

To show that there is no dependency also on the HDFS block size and therefore on the number of blocks that the technique produces, we perform an additional experiment where we consider a collection of datasets with diagonal distribution from 20 to 200 GB. For each dataset, the master files of four partitioning techniques are generated, and quality measure $Q1$ (i.e., total area) is computed. The number of blocks generated by the different techniques is changing, since the dataset size changes while the HDFS block size is fixed to 128 MB. Results are shown in Table 9. Notice that again the best technique is almost the same one (STR): The only case in which it is not corresponds to a size of 50 GB. However, in this case STR is very close to the best technique (R*-Grove) in terms of quality with a difference of only 4%. This confirms that we can train the model considering medium-size synthetic datasets without sacrificing the accuracy of the model.

5.7 Performance of the Summarization Phase

This section discusses the performance of the proposed approach for generating the summarization of each dataset, which is particularly important, since this computation has an impact on both the generation of the training set and the application phase. Notice that the first is only applied once, while the second is at work when the solution is operative.

We focus on the algorithm that computes the master files, and hence the quality metrics, on one side and the procedure that generates the histogram of a dataset on the other side. The first one is used only for generating the training set, and the second one is used also in the application phase.

For the latter, we compute the histograms using Spark as further explained in Teference [9], while to compute the master files for the six partitioning techniques we use our optimized algorithm, which is mentioned in Section 3.3 to correctly compute the six collections of master files in one job without physically partitioning the data.

In both cases, we consider as baseline approach the algorithm that physically partitions the data using the six partitioning techniques and then collects the master files from the outputs and determines the best technique by considering five quality metrics.

Figure 18 shows the efficiency of computing the histogram and the master files. It is clear that our method of generating the master files is much faster than the baseline method; this allow us to save a lot of time when producing the training set for synthetic datasets. Notice that we only compute master files for training purpose, where we compute the label for a dataset by determining the best partitioning option based on master files. In the application phase, we only need to compute the histogram or skewness features of the given dataset to predict the best partitioning

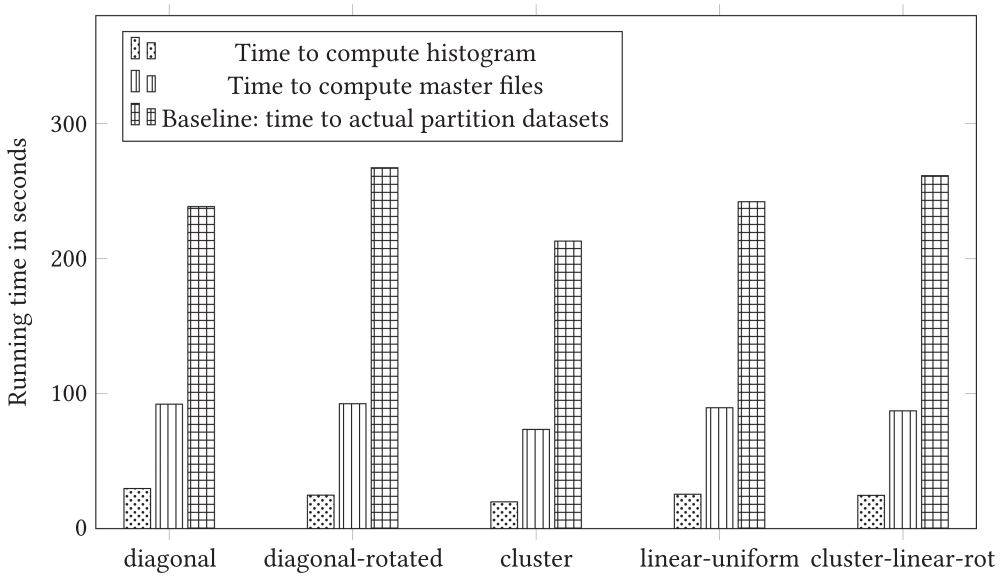


Fig. 18. Summarization performance.

technique. Figure 18 also indicates that the time to compute histogram of a dataset is very small when compared to the time to compute master files. This promises that if we have a good-enough trained model, we can quickly predict the best partitioning option instead of actual computing the master files for all techniques to determine the best one.

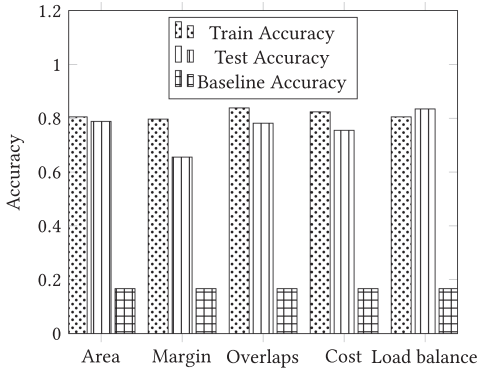
5.8 Training Data with Skewed Shapes

In this section, we study the effect of using non-point training data. More precisely, in the previous experiments the generated synthetic datasets contain rectangles that, considering the reference space, are relatively small, since they have to represents real objects like buildings or road segments compared to the extension of a state or continent. The goal is to explore whether a non-point dataset, i.e., dataset containing big and oblong rectangles, would enrich the model by extending the dataset characteristics. Figure 4 illustrates an example how the new datasets look like. We generate a total of 60 new datasets that follow the six distributions illustrated in Figure 3.

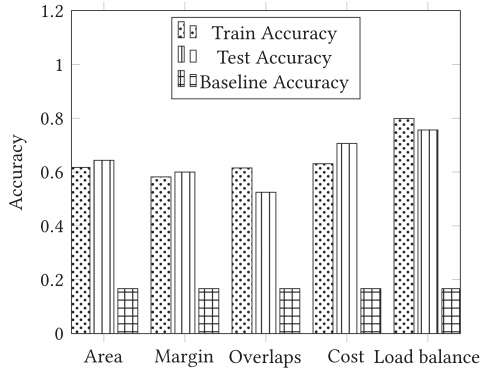
Figure 19 shows the results of the model when the input dataset contains a mix of points and rectangular datasets. Comparing these results with the results in Figure 13, we have two observations. First, the accuracy improves when adding the oblong rectangle datasets to the training sets, which affirms that non-point data enriches the training set by adding new characteristics. This is especially true when testing on real data (Figure 19(c) and (d) as compared to Figure 13(c) and (d)). Second, the gap between the histogram-based summarization and the fractal-based skewness measure summarization is reduced after using the non-point dataset. This shows a promise in deep learning being more efficient than the hand-crafted skewness measures provided that we can generate training datasets with diverse distributions and characteristics.

6 RELATED WORK

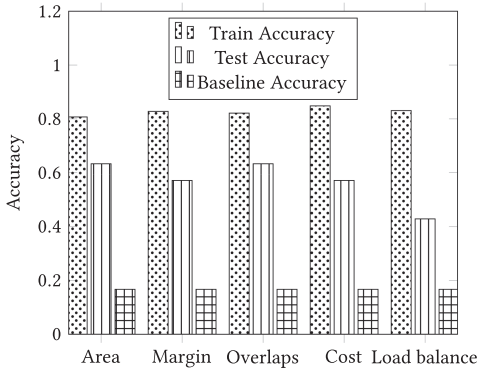
In this section, we review the related work in literature in three categories: spatial partitioning, data summarization, and deep learning.



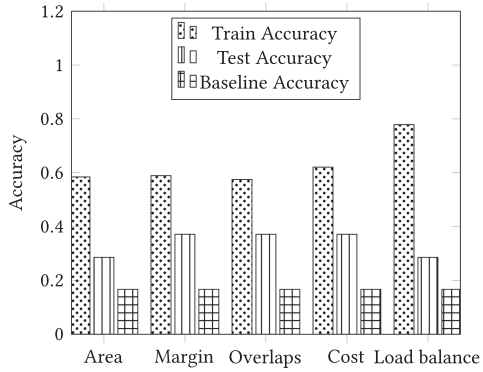
(a) Histogram summarization with test on **synthetic** data



(b) Skewness summarization with test on **synthetic** data



(c) Histogram summarization with test on **real** data



(d) Skewness summarization with test on **real** data

Fig. 19. Experiments on training data with skewed shapes.

6.1 Spatial Partitioning

Spatial partitioning is an essential operation in all big spatial data frameworks [14]. Regardless of the underlying architecture, e.g., disk-based or memory-based, data partitioning is essential to scale out to multiple machines. SpatialHadoop [13] proposed the idea of sampling-based partitioning in which a sample is used to estimate the data distribution and then a partitioning is applied to the big dataset in parallel. This idea was generalized to seven partitioning techniques including Grid-based, R-tree-based, Quad-tree-based, and space-filling-curve-based techniques [11]. Other systems follow a similar approach such as Scala-GiST [23], SATO [33], GeoSpark [39], and Simba [38]. AQWA [2] uses an adaptive histogram rather than a sample to summarize the data and query workload for data partitioning. In Reference [24], a Voronoi-diagram-based partitioning technique is proposed to solve the kNN-join operation. R*-Grove [35] is another spatial partitioning technique that extends the R-tree family for big spatial data systems. Other research work reuses some of these partitioning techniques to address other spatial analytic operations such as computational geometry operations [12] and visualization [16].

This article does not propose a new partitioning technique; rather, it proposes a framework that can suggest one of these partitioning techniques based on the input data distribution and analytic operation requirements.

6.2 Data Summarization

Many statistical techniques are used in data processing systems to provide a summarized description of a dataset, for instance, through a sample, a histogram, or a distribution model. These descriptors, often called sketches, are used to speed up the query processing by providing approximated answers based on them [10, 29–31]. One of their main uses in spatial big data analysis can be the estimation of selectivity for a join operation. The two sketching techniques that are relevant to this article can be classified into two main categories: sampling-based methods and histogram-based methods. Sampling-based methods are the basis of most existing spatial partitioning technique available in big data systems, like SATO [33], SpatialHadoop [11, 13], ScalaGiST [23], and Simba [38]. A histogram-based technique was employed by AQWA [2] to provide an adaptive partitioning technique for big spatial data based on query workload. The histogram is used to summarize the query workload, which is then used to adaptively partition the data. In general, histogram-based methods are shown to be superior for accurate spatial selectivity estimation [1, 26], and some attempts have been made to use them to answer range queries in constant time [9, 20].

This article uses histogram-based techniques to summarize the data into a fixed-size vector. Unlike AQWA [2], which used Euler histogram, this article also uses skewness measures based on these histograms including Moran's Index and box counting [6, 8].

6.3 Deep Learning

With the rise of deep learning, more research work aim at utilizing it improving decisions and recommendations such as visualization recommendation [21], query optimization [34]. One of the notable works is the learned index structures [22], which replaces the complex index structures for datasets with certain characteristics with a small neural network model and an auxiliary data structure. Similarly, there has been some work on learning locality sensitive hashing (LSH) to build approximate nearest neighbor (ANN) indexes [37]. In this work, we do not aim to replace existing methods but to alleviate the choice between existing ones using deep learning.

7 CONCLUSION

This article explores the use of deep learning techniques to choose an appropriate spatial partitioning method. It formally defines partitioning techniques, quality metrics, and the partitioning selection problem that aims at choosing the partitioning technique that will maximize a given quality metric for a dataset. The proposed framework runs in two phases, training and application. The training phase builds a deep learning model by generating synthetic datasets of diverse distributions. It uses these synthetic datasets to train a model by choosing the best partitioning technique for each one. To allow the deep learning model to work with a variable size dataset, we choose and contrast two summarization techniques termed fractal-based and histogram-based techniques. The application phase uses this model to choose the best spatial partitioning technique. We build a prototype of this framework that uses six partitioning techniques and four different quality metrics. The experimental results show up to 87% accuracy of the proposed model in recommending the best partitioning technique. We also found that the histogram-based summarization is more efficient for synthetic data while the fractal-based techniques are more efficient with real data. This suggests that we can increase the size and diversity of the training data to achieve a higher accuracy with histogram-based technique. In summary, the results show that deep learning can be used to catch the spatial data distribution in an efficient and concise way.

REFERENCES

[1] A. Aboulnaga and J. F. Naughton. 2000. Accurate estimation of the cost of spatial selections. In *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*. IEEE, 123–134.

[2] Ahmed M. Aly, Ahmed R. Mahmood, Mohamed S. Hassan, Walid G. Aref, Mourad Ouzzani, Hazem Elmeleegy, and Thamir Qadah. 2015. AQWA: Adaptive query workload aware partitioning of big spatial data. *Proc. VLDB Endow.* 8, 13 (2015), 2062–2073.

[3] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*. ACM, 322–331.

[4] Norbert Beckmann and Bernhard Seeger. 2009. A revised R*-tree in comparison with related index structures. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'09)*. ACM, 799–812.

[5] Alberto Belussi and Christos Faloutsos. 1998. Self-spacial join selectivity estimation using fractal concepts. *ACM Trans. Inf. Syst.* 16, 2 (1998), 161–201.

[6] Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2018. Detecting skewness of big spatial data in SpatialHadoop. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'18)*. ACM, 432–435.

[7] Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2020. Cost estimation of spatial join in SpatialHadoop. *GeoInformatica* (2020), 1–39. DOI: https://doi.org/10.1007/s10707-020-00414-x

[8] Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2020. Skewness-based partitioning in SpatialHadoop. *ISPRS Int. J. Geo-Inf.* 9, 4 (2020), 201. DOI: https://doi.org/10.3390/ijgi9040201

[9] Harry Chasparis and Ahmed Eldawy. 2017. Experimental evaluation of selectivity estimation on big spatial data. In *Proceedings of the 4th International ACM Workshop on Managing and Mining Enriched Geo-Spatial Data*. ACM, 8:1–8:6.

[10] Graham Cormode. 2011. Sketch techniques for approximate query processing. In *Foundations and Trends in Databases*. NOW, 7.

[11] A. Eldawy, L. Alarabi, and M. F. Mokbel. 2015. Spatial partitioning techniques in SpatialHadoop. *Proc. VLDB Endow.* 8, 12 (August 2015), 1602–1605.

[12] Ahmed Eldawy, Yuan Li, Mohamed F. Mokbel, and Ravi Janardan. 2013. CG_Hadoop: Computational geometry in MapReduce. In *Proceedings of the 21st SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*. ACM, 284–293.

[13] Ahmed Eldawy and Mohamed F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *Proceedings of the 31st IEEE International Conference on Data Engineering '15)*. IEEE, 1352–1363.

[14] Ahmed Eldawy and Mohamed F. Mokbel. 2016. The era of big spatial data: A survey. *Found. Trends Databases* 6, 3–4 (2016), 163–273. DOI: https://doi.org/10.1561/1900000054

[15] A. Eldawy and M. F. Mokbel. 2017. *Spatial Join with Hadoop*. Springer International Publishing, Cham, 2032–2036.

[16] Ahmed Eldawy, Mohamed F. Mokbel, and Christopher Jonathan. 2016. HadoopViz: A MapReduce framework for extensible visualization of big spatial data. In *Proceedings of the 32nd IEEE International Conference on Data Engineering, (ICDE'16)*. IEEE, 601–612.

[17] Sattam Alsubaiee et al. 2014. AsterixDB: A scalable, open source BDMS. *Proc. VLDB* 7, 14 (2014), 1905–1916. DOI: https://doi.org/10.14778/2733085.2733096

[18] Saheli Ghosh, Tin Vu, Mehrad Amin Eskandari, and Ahmed Eldawy. 2019. UCR-STAR: The UCR spatio-temporal active repository. *SIGSPATIAL Special* 11, 2 (December 2019), 34–40. DOI: https://doi.org/10.1145/3377000.3377005

[19] Antonio Gulli et al. 2017. *Deep Learning with Keras*. Packt Publishing Ltd, UK.

[20] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. 1997. Range queries in OLAP data cubes. *SIGMOD Rec.* 26, 2 (1997), 73–88.

[21] Kevin Zeng Hu, Michiel A. Bakker, Stephen Li, Tim Kraska, and César A. Hidalgo. 2019. VizML: A machine learning approach to visualization recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI'19)*. ACM, 128.

[22] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD'18)*. ACM, 489–504.

[23] Peng Lu, Gang Chen, Beng Chin Ooi, Hoang Tam Vo, and Sai Wu. 2014. ScalaGiST: Scalable generalized search trees for mapreduce systems [Innovative Systems Paper]. *Proc. VLDB Endow.* 7, 14 (2014), 1797–1808.

[24] Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. 2012. Efficient processing of k nearest neighbor joins using MapReduce. *Proc. VLDB* 5, 10 (2012), 1016–1027. DOI: https://doi.org/10.14778/2336664.2336674

[25] P. A. P. Moran. 1950. Notes on continuous stochastic phenomena. *Biometrika* 37, 1 (1950), 17–23.

[26] Viswanath Poosala and Yannis E. Ioannidis. 1997. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 23rd International Conference on Very Large Data Bases*. Morgan Kaufmann, San Francisco, CA, 486–495.

[27] Sebastian Raschka. 2018. Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. arXiv:cs.LG/1811.12808

[28] Salman Ahmed Shaikh, Komal Mariam, Hiroyuki Kitagawa, and Kyoung-Sook Kim. 2020. GeoFlink: A framework for the real-time processing of spatial streams. *arXiv preprint arXiv:2004.03352* 1, 1 (2020), 1.

[29] A. B. Siddique, Ahmed Eldawy, and Vagelis Hristidis. 2019. Euler++: An improved selectivity estimation for rectangular spatial records. In *Proceedings of the IEEE Big Spatial Data Workshop*. IEEE, USA, 1.

[30] A. B. Siddique and Ahmed Eldawy. 2018. Experimental evaluation of sketching techniques for big spatial data. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC'18)*. ACM, USA, 522.

[31] Abu Bakar Siddique, Ahmed Eldawy, and Vagelis Hristidis. 2019. Comparing synopsis techniques for approximate spatial data analysis. *Proc. VLDB Endow.* 12, 11 (2019), 1583–1596.

[32] M. Tang, Y. Yu, W. G. Aref, A. R. Mahmood, Q. M. Malluhi, and M. Ouzzani. 2019. LocationSpark: In-memory distributed spatial query processing and optimization. *CoRR*.

[33] Hoang Vo, Ablimit Aji, and Fusheng Wang. 2014. SATO: A spatial data partitioning framework for scalable query processing. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, New York, NY, 545–548.

[34] Tin Vu. 2019. Deep query optimization. In *Proceedings of the 2019 International Conference on Management of Data*. ACM, 1856–1858.

[35] Tin Vu and Ahmed Eldawy. 2018. R-Grove: Growing a family of R-trees in the big-data forest. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'18)*. ACM, 532–535.

[36] Tin Vu, Sara Migliorini, Ahmed Eldawy, and Alberto Bulussi. 2019. Spatial data generators. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Spatial Gems (SpatialGems 2019)*. ACM, 1.

[37] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. 2016. Learning to hash for indexing big data – survey. *Proc. IEEE* 104, 1 (2016), 34–57. DOI: https://doi.org/10.1109/JPROC.2015.2487976

[38] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba: Efficient in-memory spatial analytics. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, New York, NY, 1071–1085.

[39] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2015. GeoSpark: A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 70:1–70:4.

Received May 2019; revised February 2020; accepted May 2020