Chapter

JavaScript Implementation of Scagnostics and Its Applications

Vung Pham and Tommy Dang

Abstract

Scagnostics is a set of features that characterizes the 2D distributions in the underlying data. Various real-world applications have been using Scagnostics visual features to detect unusual bivariate data correlations. Concomitantly, many applications are required to be implemented on web platforms due to their accessibility and convenience. Therefore, this chapter discusses a recent JavaScript implementation of Scagnostics, an extension to higher dimensional data, and its applications in detecting abnormalities in bivariate and multivariate time series data. Its implementation in JavaScript supports the tremendous demand for visual features in the web environment. Likewise, its higher dimensional implementations allow generating Scagnostics features for the rapidly growing multivariate data. Finally, conventional ScagnosticsJS computations involve time-consuming algorithms, and they are sensitive to slight changes in the underlying data. Therefore, this chapter also discusses a recent attempt to tackle these issues using machine learning to estimate the Scagnostics scores.

Keywords: Scagnostics, 3D Scagnostics, nD Scagnostics, JavaScript, Visual Features for Scatterplots, Data Correlation, High dimensional Data Analysis

1. Introduction

Visualizations on the web platform are getting popular because they are likely to be viewed on various devices without making any complicated software setup. A critical aspect of data visualization is detecting and highlighting the visual features in the underlying data. Regarding visual features, Scagnostics [1] is a popular set of features that allows characterizing the distribution of the underlying data. Though its two-dimensional (2D) version is prevalent, three-dimensional (3D) and higher-dimensional (nD) Scagnostics uses are limited.

There are several reasons for this limitation. First the 2D version of Scagnostics implementations were limited in programming languages like R [2], Python [3], and Java [4]. Furthermore, there was only one attempt at implementing Scagnostics in 3D space [5], and it is also supported in Java only. These programming languages encumber the uses of these visual features for the web, which is gaining popularity due to its convenience of use. Moreover, the lack of nD version nukes their uses for multivariate data in many application domains. Examples of such areas are those that often require multivariate data for better reliability and statistically sound analysis.

This chapter discusses a recent work called ScagnosticsJS [6]. It is a Scagnostics library implemented in JavaScript to support visualizing data features on the web.

Furthermore, it is the first officially published library that extends the Scagnostics features into 3D and nD. This chapter also discusses the uses of 2D and nD versions of ScagnosticsJS in detecting abnormalities in bivariate as well as multivariate time series data, namely Outliagnostics [7] and MTSAD [8], respectively. Finally, conventional Scagnostics computations involve algorithms that are time-consuming and sensitive to slight changes in the underlying data. Thus, this chapter's last section outlines an attempt to tackle these issues using machine learning, called ScagCNN [9]. Its main idea is to train and use the learned machine learning model to predict the Scagnostics scores instead of executing the conventional algorithms.

2. 2D Scagnostics

In 2005, Wilkinson [1] proposed and implemented a set of the graph-theoretic summaries of two-dimensional scattered point data, called Scagnostics. Since then, Scagnostics has been used in various application domains, such as high-dimensional time series analysis [8], clustering of scatter plots [4], and abnormality detection [7], to name but a few. This section summarizes the nine Scagnostics measures and their visual characteristics. We also refer interested readers to the original paper [1] for further details. These scores are broadly categorized into density, shape, and association measures.

There are five Scagnostics measures that are computed using density geometric features, namely: *outlying*, *skewed*, *sparse*, *clumpy*, and *striated*. They are characterized by the distribution of the edge lengths of the minimum spanning tree (MST) built on the 2D scatter points. Next, there are three Scagnostics measures that represent the shape geometric feature of scattered point data, namely *convex*, *skinny*, and *stringy*. Besides the MST, the shape measures also leverage the alpha (A) and convex (H) hulls built on top of the scattered points. Lastly, the association measure represents the symmetric measure of association between the two variables involved. The only one Scagnostics score in this category is *monotonic* score. Scagnostics uses the squared Spearman correlation coefficient of the 2D data to compute this score. We refer interested readers to the original work [10] regarding the formal definitions and how to compute these 2D Scagnostics scores.

There are several Scagnostics implementations in different programming languages such as R [2], Python [3], and Java [4]. These implementations are for 2D Scagnostics, and there is only one attempt to implement Scagnostics in application to 3D data [5]. ScagnosticsJS [6] is the first officially published Scagnostics implementation in JavaScript. It also extends the 2D Scagnostics measures for 3D and higher dimensional (nD) data points. The remained sections of this chapter discuss ScagnosticsJS implementations, its extensions, and its applications.

3. ScagnosticsJS: 2D, 3D, and nD Scagnostics for the web

This section describes the 2D, 3D, and nD Scagnostics implementation called ScagnosticsJS [6]. It is a library for Scagnostics in JavaScript, which is gaining popularity for *visualization for the web* [11]. Besides 2D Scagnostics, it is also the first officially published extensions of Scagnostics to 3D and nD. **Figure 1** shows the nine Scagnostics measures and their exemplar plots in 2D (a), 3D (b), and nD (c). The heatmaps next to these plots show the corresponding Scagnostics scores computed using ScagnosticsJS. Notably, ScagnosticsJS gives high scores (the highlighted, black bordered cells at the diagonals of the heatmaps) for the Scagnostics types that the scatterplots were generated to flag.

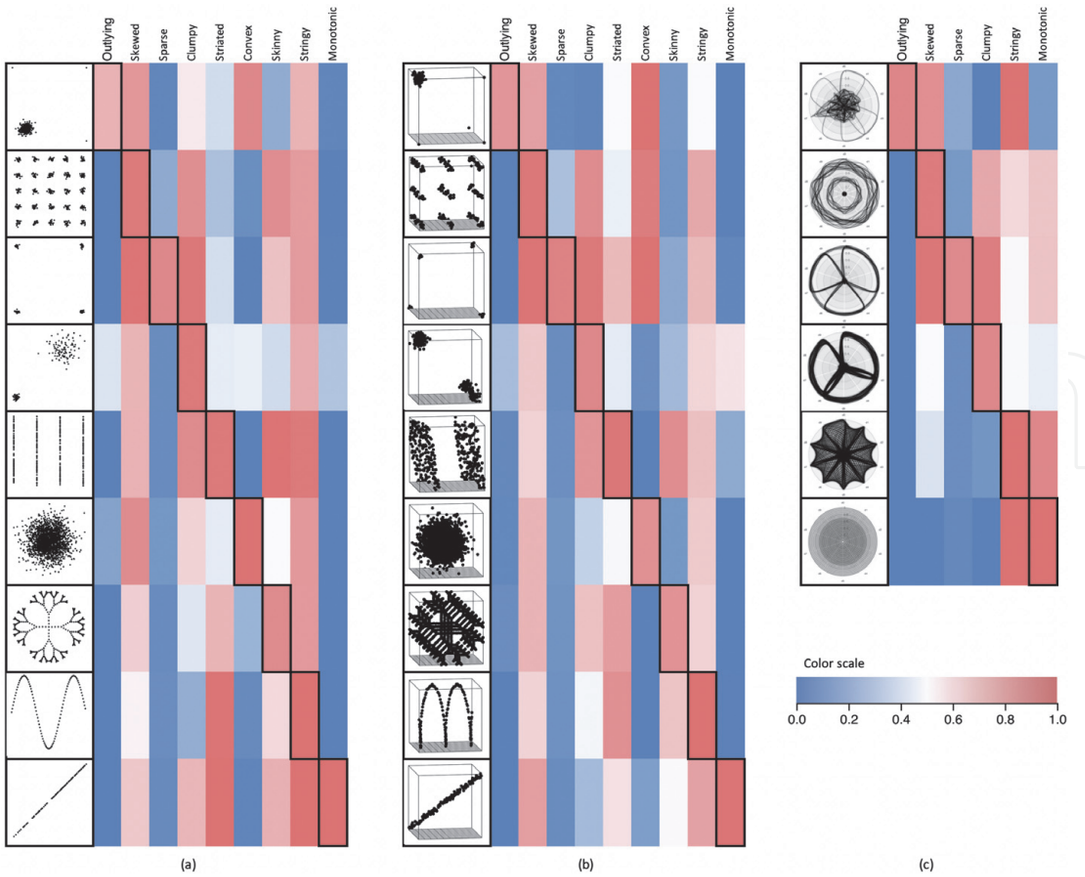

Figure 1.
Scagnostics measures and their exemplar plots in 2D (a), 3D (b), and nD (c). The heatmaps show corresponding Scagnostics scores computed using ScagnosticsJS. Highlighted, black-bordered cells at the diagonals indicate that ScagnosticsJS flags high Scagnostics scores for their corresponding typical Scagnostics plots.

3.1 Binning

The Scagnostics computation starts with the binning process. This step reduces the computation expense when handling a large number of data points and allows more stable Scagnostics computations. Rectangular binning is a simple and popular binning method in data science in general. However, hexagon and leader binning algorithms are two commonly used methods for Scagnostics computation. Though both algorithms are similar in terms of time complexity, leader binning has its advantage of preserving the underlying data's original shape. Contrariwise, hexagon binning does not work well with Box Plot Rule [7] because the distance between neighboring hexagons is always the same due to the binning shape and their consecutive arrangement.

Figure 2 depicts the difference between leader binning (d) and its rectangular (b) and hexagon (c) counterparts on the same original data (a). Each leader's size

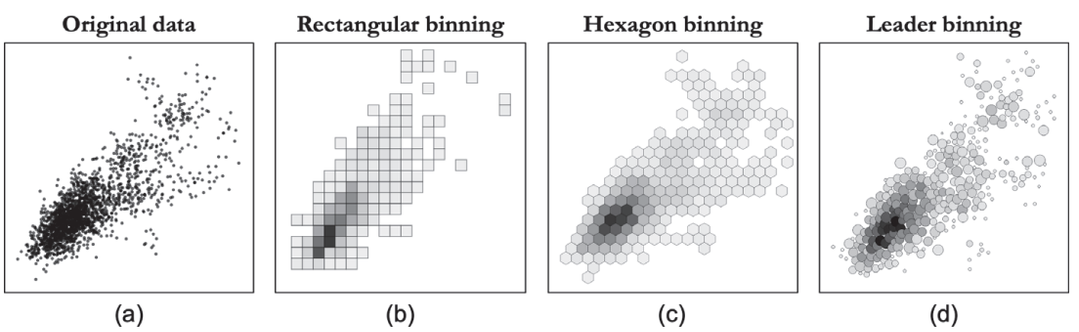

Figure 2.
Original data (a) and binning methods: Rectangular binning (b), hexagon binning (c), and leader binning (d).

indicates its coverage, while its intensity highlights the number of points that fall into that bin. Each binning algorithm has its pros and cons depending on the data and the analysis task, so ScagnosticsJS provides the flexibility to choose either hexagon or leader binning in its 2D version. It is hard to find an appropriate shape for the hexagon in nD, but a hyper-sphere is an appropriate representation in nD equivalent to a circle in 2D. Therefore, ScagnosticsJS uses leader binning for its nD implementation.

3.2 ScagnosticsJS 2D implementation

The 2D ScagnosticsJS implementation includes several intermediate computation stages as depicted in **Figure 3**. They include 1) normalization, 2) binning, 3) generating triangulation, 4) computing MST, 5) finding degree 1 and 2 vertices, and 6) finding convex and concave hulls. Since 2D Scagnostics implementation is popular, we refer interested readers to these original papers [1, 6] for further details regarding these stages' implementations.

3.3 ScagnosticsJS 3D implementation

Our 3D implementation of Scagnostics in ScagnosticsJS bases mainly on how corresponding geometries transferred from 2D space into 3D space. The concept of the MST is exactly the same as that in 3D using Euclidean distance. Obviously, the *outlying*, *skewed*, *stringy*, and *clumpy* measures use the distribution of the lengths of the MST's edges. Thus, their 3D implementations can be naturally adapted from 2D. Additionally, *monotonic* score in 3D is the partial correlations of the three variables: $c_{monotonic} = \max\left[\rho^2_{X,Y|Z}, \rho^2_{X,Z|Y}, \rho^2_{Y,Z|X}\right]$.

Striated patterns in 2D involve parallel and smooth (e.g., spiral) lines [1]. The equivalent *striated* patterns in 3D involves all of these 2D patterns plus the parallel planes. So, the *striated* score is revised to account the angle between adjacent planes (p) formed by every three consecutive edges (e_1, e, and e_2) of the MST. Different from [5], we need to consider $|\cos\theta| > 0.75$ instead of $\cos\theta < -0.75$ as the 2D version. Therefore, the 3D *striated* score is now computed as:

$$c_{striated} = \frac{1}{|V|}\sum_{v \in V^{(\geq 2)}} I\left(|\cos\theta_{p(e,e_1)p(e,e_2)}| > 0.75\right) \quad (1)$$

where $V^{(\geq 2)} \subseteq V$ are vertices of degree ≥ 2. The $\cos\theta_{p(e,e_1)p(e,e_2)}$ is calculated as the dot product of the unit normal vectors of the two planes $p(e, e_1)$ and $p(e, e_2)$. These unit normal vectors are, in turn, computed using the cross products of the vectors made of source and target nodes of e and e_1 for the first plane and e and e_2 for the second plane.

Figure 4 shows the synthesized samples generated to justify the use of $|\cos\theta| > 0.75$ instead of $\cos\theta < -0.75$. Specifically, these samples include two

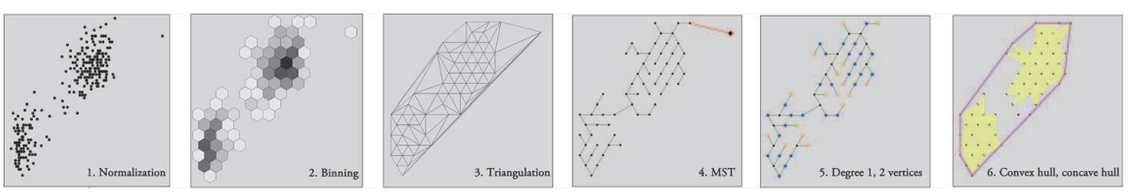

Figure 3.
Main algorithms used in 2D Scagnostics computation: 1) normalization, 2) binning, 3) triangulation, 4) MST (with a red outlier), 5) vertices with degree 1 (orange) and 2 (blue), and 6) convex and concave hulls.

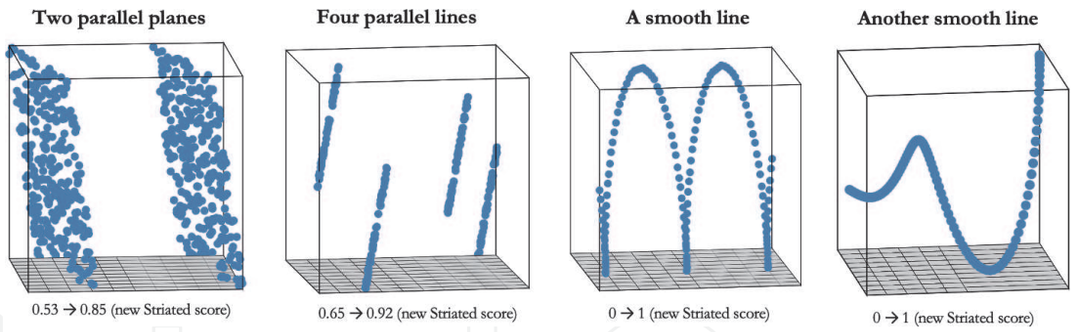

Figure 4.
Comparing striated scores using $\cos\theta < -0.75$ *vs.* $|\cos\theta| > 0.75$ *thresholds for two parallel planes, four parallel lines, a smooth line, and another smooth line to test the generality of the proposed formula.*

parallel planes, four parallel lines, and two smooth lines (the last two cases). The *striated* scores for the first two cases are relatively similar and are reasonable. However, in the third case, using $\cos\theta < -0.75$ threshold leads to *striated* score of 0 versus 1 when using $|\cos\theta| > 0.75$ threshold. As of the *striated* pattern definition, this score of a smooth line should be 1. We also tested the proposed change to other types of smooth lines to ensure that this formula is generalized to measure the smooth lines' striated visual feature. The last case is another example of such many smooth lines that we had tested on.

The 2D *c onvex* and *skinny* scores leverage the perimeters and areas of the convex and concave hulls built on top of the underlying data. In 3D, the 2D lines are equivalent to planes, and the α radius for the circle used to calculate the 2D alpha hull is that for the sphere in 3D used to calculate the 3D alpha hull. Therefore, 3D version convex and skinny scores use the equivalent surface areas and volumes of these shapes in 3D, instead of perimeters and areas as in 2D. These scores are formally defined as:

$$c_{convex} = volume(A)/volume(H) \tag{2}$$

$$c_{skinny} = 1 - \sqrt[6]{36\pi}\sqrt[3]{volume(A)}/\sqrt{surfacearea(A)} \tag{3}$$

where H and A are the convex and alpha hulls in 3D space, respectively. Also, the constant $\sqrt[6]{36\pi}$ is to assure that a sphere has $c_{skinny} = 0$.

The area of a 3D hull is the sum of areas of all triangles of that hull. These triangles are the faces of the hull as the result of the hull computation. Also, ScagnosticsJS uses the algorithms from Robert Nürnberg [12] to efficiently compute the volume of a 3D hull. Algorithm 1 summarizes the steps for computing the volume of a 3D hull from its faces. It is worth noting that this algorithm works for both convex and concave hulls.

Algorithm 1 Compute the volume of 3D hull from its set of faces.

1: **procedure** COMPUTEVOLUME(*faces*).
2: initialization: n = number of faces, $volume = 0$, $i = 0$.
3: **while** $i < n$ **do**:
4: $triangle_i = faces[i]$
5: $vector_i$ = one vertex of $triangle_i$.
6: $\hat{n}$ = normal vector of $triangle_i$.
7: $volume = volume +$ dot product of $vector_i$ and $\hat{n}$
8: $i = i + 1$
9: **return** *volume*.

Lastly, we defer the discussion regarding the computation of *sparse* score to the next section. The reason is that this measurement deserves further discussions and is generalized from 2D to 3D and naturally to nD versions.

3.4 ScagnosticsJS nD implementation

Like the 3D implementation, the nD version depends on how geometries in 2D and 3D spaces are translated into nD. Specifically, regarding building the MST in nD space, besides Euclidean distance, one should also consider using other metrics such as Manhattan distance metric (L_1 norm) or L_k norm where k is a fraction. Asides from the distance metric consideration, the MST computation remains the same, so do the computations of the *outlying*, *skewed*, *sparse*, *clumpy*, and *stringy* scores. Also, ScagnosticsJS uses maximum partial monotonicity among all pairs of variables as the *monotonic* score in nD.

With values normalized to the range [0, 1] as described in the 2D implementation section, in nD space, the maximum distance between any two points could get up to $n^{1/k}$, where n is the number of dimensions, and k is the value for the distance metric L_k used. For instance, in case of Euclidean distance ($k = 2$), this distance is $\sqrt{n}$. This case is an extreme one because, at most, there is only one edge length with such value. However, the fact is that the higher the number of dimensions, the higher the possibility that the MST edges get longer than 1. Therefore, using q_{90} of the MST edge length distributions as the *sparse* score does not generalize to 3D or nD versions.

Figure 5 shows 6D, 10D, and 16D sets of data points synthetically generated to illustrate this circumstance. If we use the q_{90} as a sparse measure, the three plots have sparse scores as 1.76, 2.16, and 2.80, respectively. There are two issues with these scores. First, they are out of the Scagnostics score range (0 to 1). Second, the three synthesized plots are visually similar, and they only differ in numbers of dimensions. In other words, the higher sparse scores of the higher dimensional plots are due to the higher number of dimensions they have, not due to their intrinsic visual features. Therefore, ScagnosticsJS proposes the sparse score as:

$$c_{sparse} = q_{90} / \sqrt{\left\lfloor \frac{2 \times n}{3} \right\rfloor} \tag{4}$$

where n is the number of dimensions, and we use the floor operation ($\lfloor\rfloor$) because MST always selects the lower distances first. The numerator 2 is due to the pairwise distance between points. The denominator 3 is the requirement that we would like to have at least 3 MST edges with lengths greater than or equal to the q_{90}. Also, this is compatible with the cases $n = 2$ in 2D implementation. This correction

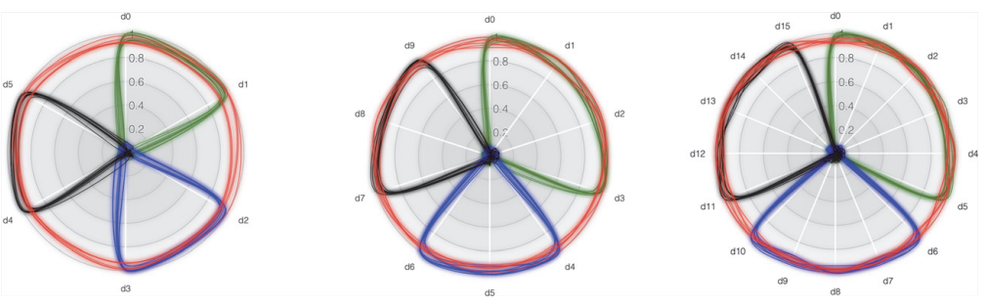

Figure 5.
Synthesized 6D, 10D, and 16D plots with high sparse scores: 1.76, 2.16, and 2.80 if we use the q_{90} measure, though they are visually similar. Radar chart, in this case, is one of many ways to represent multivariate data, and line colors represent different classes.

factor normalizes the sparse scores for the scatter plots in **Figure 5** into approximately 0.88 (0.880, 0.882, and 0.885, respectively) in these three cases.

Currently, ScagnosticsJS does not implement the nD *convex* and *skinny* scores due to their limited utility [13]. Furthermore, these scores depend on the convex/alpha hull in nD space, and the computations of these shapes pose performance constraints. Specifically, even with an approximation approach such as one from [14], the complexity of hull calculation is still $\mathcal{O}\left(N^2 m^{3/2} log\ \frac{m}{\epsilon_0}\right)$, where m is close to the number of vertices of approximation and ϵ_0 is the maximum error. The time complexity makes it impractical to incorporate these scores in the current nD version.

Similarly, current ScagnosticsJS version does not provide the implementation for *striated* score in nD space. There are several reasons for this. First, calculating this score involves computing the angle between every two consecutive MST edges or two planes formed by every three consecutive MST edges in 2D and 3D cases, respectively. This calculation further involves the cross-product of two vectors. This cross-product does not exist in the space with dimensions higher than three. In other words, there are infinitely many unit vectors orthogonal to any given two. The second reason is also about utility. Specifically, lines and planes can be found in higher-dimensional space, but there is not often much reason to use them [15].

Interested readers can refer to the Github page of ScagnosticsJS, at https://idatavisualizationlab.github.io/ScagnosticsJS, to explore and learn how to use this library in readers' application domains. The next section describes two typical recent applications of this library to demonstrate the uses of 2D and nD Scagnostics to extract visual features for visualizations on the web platform.

4. Applications of ScagnosticsJS

JavaScript implementation of Scagnostics facilitates the use of these measures on the web. This section describes two recent works called Outliagnostics [7] and MTSAD [8] as the typical applications of 2D and nD ScagnosticsJS, respectively.

4.1 Visualizing temporal discrepancies in outlying signatures of data entries

Outliagnostics [7] is a recent, typical application of 2D ScagnosticsJS. It is an application of this library to analyzing 2D temporal data concentrating on detecting data entries that are significant in contributing to the outlying score of a scatterplot. Its prototype also supports interactive explorations of abnormalities in large time series. It uses ScagnosticsJS to calculate how much a data point adds to the outlying score of the underlying scatter points as a whole at every time point. In some cases, outliers can be detectable using one-dimensional data. However, in many other cases, they are only detectable in multidimensional space. Therefore, instead of using conventional approaches such as Box Plot Rule to find outliers in one-dimensional data, ScagnosticsJS allows determining the outliers using two-dimensional data points.

Figure 6 demonstrates the use cases of 2D outlying detection where these outliers in any marginal projections are indiscernible. **Figure 6 (a)** is the scatterplot of international debt data (*debt* vs. *population*) in 2017: Each data point is a country in the example scatterplot. One can rely on the *debt* axis alone to determine that Pakistan and China are outliers, or the *population* axis alone to discern China and India as outliers. There are similar cases in **Figure 6 (b)** for the New York stock exchange in January 2011 (*price* vs. *volume*), such as BAC, AAPL, GOOG. However,

it is relatively hard to tell if NFLX is outlying using either one of the axes in this scatterplot. Similarly, it is even harder to tell if Iraq, El Salvador, or Iran in **Figure 6 (c)** using either *Female* axis or *Male* axis alone in this *Life expectancy in 1982* scatterplot. However, they are visually outliers if both axes are taken into account.

While analyzing the outlying impact of data points to the overall scatterplot outlying score, besides the outliers, Outliagnostics also considers "inlying" data points. Specifically, it defines an "inlier" as an observation that lies in the interior of statistical distribution, and its absence makes the detection of other outliers easier or possible. For instance, *A* and *B* are two data entries in a distribution. *A* is an inlier if removing *A* makes *B* an outlier. Outliagnostics's approach to identifying the outlying contribution of an individual data point to its overall dataset is to compute the difference of the outlying scores while having and not having that particular data point in the underlying dataset. This method is called the *leave-one-out* approach.

This *leave-one-out* approach allows estimating the outlying contributions of individual data points to the overall scatterplot as a whole. This approach is computationally intensive. However, Outliagnostics leverages binning and parallel computing to achieve a near-linear computation complexity concerning the size of a dataset. Specifically, the leave-one-out approach is "selective" because it only leaves out singleton bins. The reason is removing an observation from a dense bin will not impact the Scagnostics outlying scores.

Figure 7 shows the main components of Outliagnostics. On the left (a) is a control panel that shows color legends and allows users to set various display properties such as ordering fields or searching for items, in case needed. The top bar (b) represents the time series going from left to right. Right below that time series bar is the set of thumbnails (c) for the 2D data represented as scatterplots over time. These scatterplots' backgrounds represent their outlying scores (red background means higher outlying score while the blue one indicates low outlying score).

The thumbnails below the tag clouds (d) visualize the five data entries with the highest contributions on the overall outlying score (increasing or reducing). The text colors in these tag clouds designate the inlying (green) or outlying (purple) contributions of the corresponding entries. Also, Outliagnostics provides a customized box-plots view (e), which gives an overview of the outlying and inlying information of the scatterplots at individual time steps. A stream graph overlayed on top of these box-plots shows how these outlying/inlying values evolve.

Finally, Outliagnostics also provides a visualization section called instance profiles, as shown in **Figure 7 (f)**. This section allows users to investigate how outlying/inlying values evolve at the individual instance level. Specifically, each item has a base-line representing the outlying score of the overall scatterplot. Suppose at a time

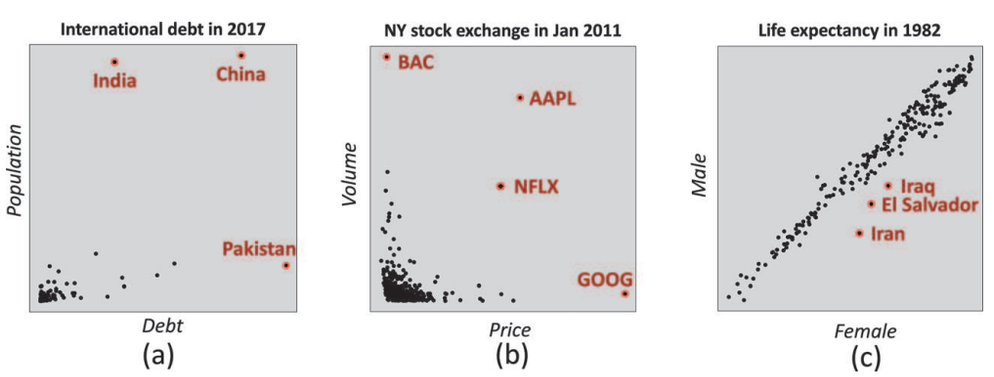

Figure 6.
Examples of two-dimensional outliers that might not be detectable in individual dimensions: (a) international debt, (b) New York stock exchange, and (c) life expectancy data from the World Bank [7].

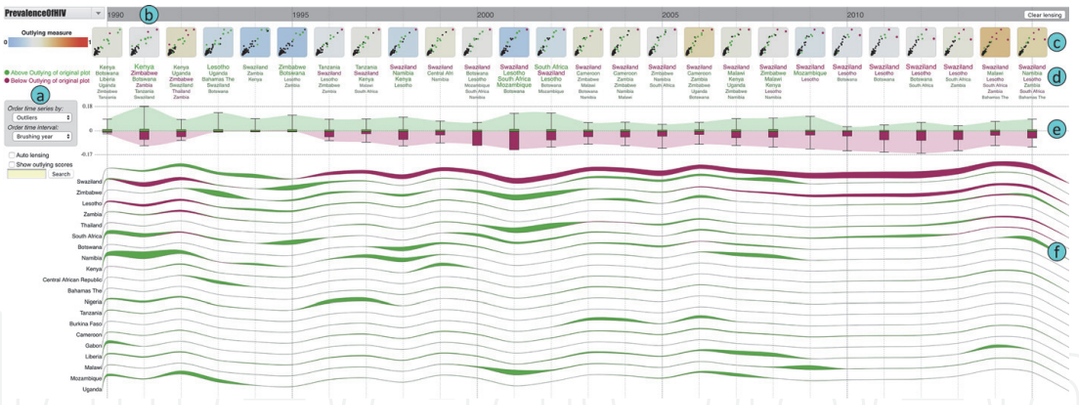

Figure 7.
Outliagnostics visualization components: (a) control panel, (b) lensing area, (c) scatterplot series, (d) top countries clouds, (e) customized outlying boxplots, and (f) outlying profiles [7].

point the instance contributes more to the inlying/outlying score. In that case, there are green streams above this base-line or purple streams below it to represent the changes of the outlying scores when leaving the current data item out, correspondingly. The streams' heights denote the inlying/outlying difference when this specific instance is present or absent from the scatterplot. The use of stream graphs for these scores allows the users to observe how the outlying scores evolve for a particular instance.

Figure 8 depicts a use-case of Outliagnostics applies to the World Life Expectancy dataset retrieved from UIC repository [16]. This dataset is the life expectancy (male vs. female) from 263 countries worldwide in 56 years. The time-series boxplots and the outlying/inlying streams highlight the 1970s, early 1990s, and late 1990s - early 2000s as three periods with high outlying scores. The time series shows Cambodia, Rwanda, and Sierra Leone as the three profiles that are ranked on top of the list with thicker outlying streams. In other words, they contribute more to the overall outlying scores over these time periods, correspondingly. Clicking on the thumbnails scatter plots at each of the periods shows their detailed views in boxes (a), (b), and (c), respectively. Moreover, users can mouse-over these three countries in these detailed views to inspect the male and female life expectancy for these countries. Specifically, Combodia had as low as 27 (male) and 21 (female) in 1975. These numbers are 30 and 26 for Rwanda in 1992. Similarly, They were 38 and 36 for Sierra Leone in 1998. These low values were caused by the 1978–1991

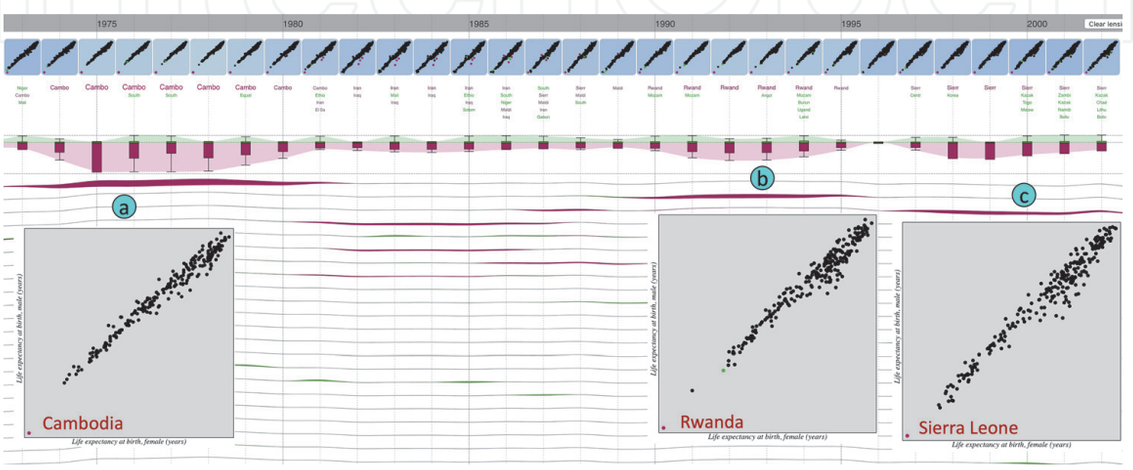

Figure 8.
Outliagnostics highlights Cambodia, Rwanda, and Sierra Leone as outliers in the 1970s (a), early 1990s (b), and late 1990s - early 2000s (c), respectively.

Cambodian Civil War, the 1990–1994 Rwanda Civil War, and the 1991–2002 Sierra Leone Civil War, respectively.

Finally, source codes, video demo, and web prototypes of Outliagnostics are available from this project's Github page at https://outliagnostics.github.io/index.html.

4.2 Multivariate time series abnormality detection and visualization

Several outlying items are not detectable using a lower number of dimensions, but they are in higher dimensional space, as discussed in Section 4.1. Therefore, this section describes another recent application of the nD version of ScagnosticsJS [6] that applies to detecting abnormalities within a multivariate time series using a web-based application, called MTSAD [8]. This application is a natural extension of Outliagnostics to the nD data. Specifically, Outliagnostics claims that several outliers are not detectable using an individual variable, but they are evident when both variables of the bivariate data are taken into account. Similarly, one may see the same argument for 3D and nD space.

Figure 9 shows the main interface of MTSAD that monitors nine critical variables in a high-performance computing center [17]. Similar to Outliagnostics, MTSAD also contains a time-line, plot preview thumbnails over time, the customized box-plots depicting how overall outlying/inlying scores evolve, the item profile sections, and a control panel with interactive options to support exploration during the abnormality detection process. The same leave-one-out strategy, as described in Section 4.1, is used to calculate the outlying contribution of an individual data instance to the overall outlying score of the whole set (i.e., of all involving data items) at a specific time step.

There are two main differences between Outliagnostics and MTSAD. First, the former uses 2D ScagnosticsJS while the latter utilizes the nD version to calculate the outlying scores for individual data-plot at each time step. The second difference is that MTSAD leverages small-multiples (radar-charts) to show the multivariate time series instead of showing scatterplot for 2D data points as in Outliagnostics. Each small-multiple visualizes the monitoring variables at a time step. Each radar-chart might potentially need to render a large number of data entries (467 computation nodes in this case). This large number of paths indicates a high rendering time and produces visually cluttering issues. Concomitantly, a higher level of data and visualization abstractions (such as grouping and visualizing a group of data instead of individual data points) gives a better overview of the data and faster rendering time [18]. Therefore, this application only displays individual inlying (green paths) and outlying (red paths) entries, detected by the nD version of ScagnosticsJS with the leave-one-out approach. It then uses k-means algorithm [19] to aggregate other

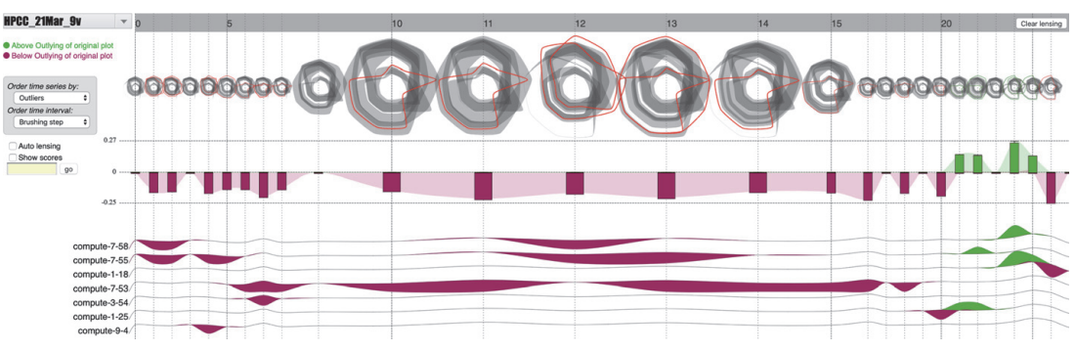

Figure 9.
MTSAD primary visualization components for a multivariate time series dataset from a high performance computing center: 1) a time-line, 2) radar-chart small-multiples, 3) overall outlying box-plots, 4) the data-entry profile, and 5) a control panel to assist the abnormality exploration interactively [8].

non-inlying/non-outlying computing nodes into clusters and visualizes them as gray bands to reduce rendering time. This approach highlights outlying/inlying observations in the underlying data, which assists the detection of abnormality.

Figure 10 shows a snapshot of MTSAD applies to monitoring nine essential variables from a high-performance computing center. The monitoring period is from 2:00 PM until 4:25 PM on March 21, 2019 (in 30 time-steps with five minutes each). **Figure 10** (a) shows a detailed view of these monitoring CPU-health-related metrics at 2:20 PM. Notably, some nodes were heated, and two of them were outlying because they have high values for temperature metrics. Also, some other nodes detected the heat increment and increased their fan speeds. Since then, these two metrics raised rapidly.

As depicted in **Figure 10** (b), By 3:50 PM, our visualization highlighted that the recorded values for these two variables were high for many computation nodes. Finally, our monitoring system could not receive data from the computation nodes at around 4:25 PM. MTSAD detected and reported this suspicious behavior to the high-performance computing center's administrators. They acknowledged the matter and clarified that the cooling system's fault operations (chilled water system) caused the issue. The monitoring system did not receive any signal by 4:25 PM because the system administrators performed an emergency shutdown to circumvent any further harm to the computation facility.

Furthermore, this use-case also suggests that outliers and inliers are equally important in our abnormality detection approach. As shown in **Figure 10** (b), the Box-Plot rule cannot detect outlying data entries at the 3:20 PM time step. Notably, at this time, *compute-1-25* and *compute-7-55* were at their extreme values (one had very low and another very high values for the tracking CPU-health metrics). The masking effect hides the outlying characteristics of these two instances. Specifically, leaving one of the two nodes out, the remained data points will have a high Scagnostics outlying score. That said, inliers allow us to detect potential outliers. Consequently, identifying the inlying measurements permits us to tackle the masking effect.

Finally, the source codes and web prototype of MTSAD application are available at https://idatavisualizationlab.github.io/V/MultiOutliers.

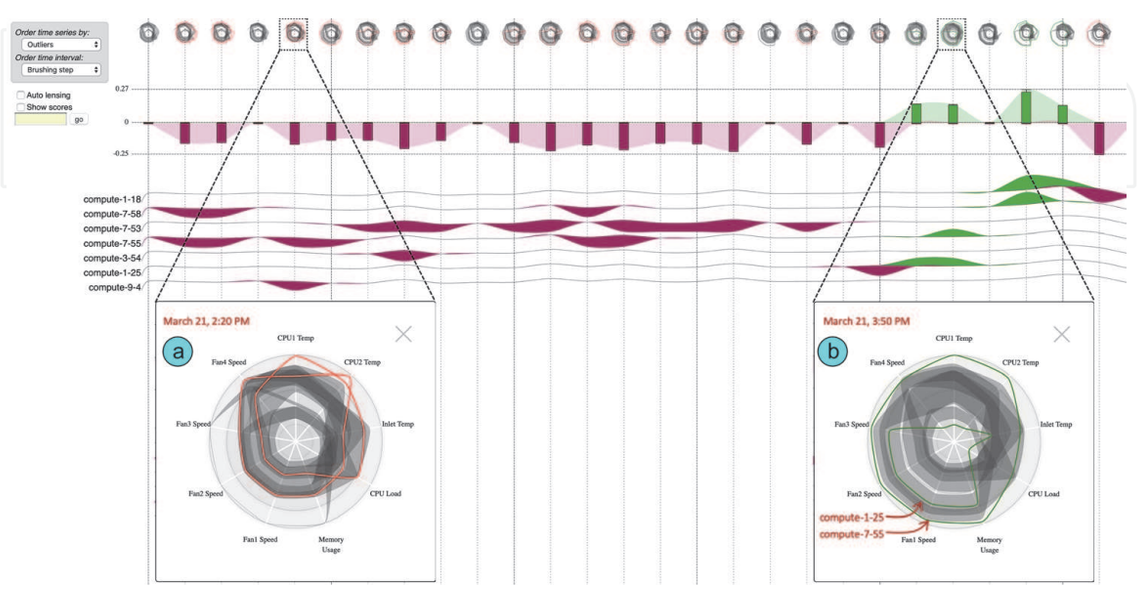

Figure 10.
MTSAD visualizations applied to monitoring essential variables at a high performance computing center. From 2:00 PM until 4:25 PM on march 21, 2019. There was a chill water issue during this period. At 2:20 PM (a), the computation nodes' temperatures increased, and a few nodes sensed the heats and pumped their fan speeds (panel a). At 3:50 PM (b), other computation nodes also increased their fan speeds after sensing the heats [8].

5. Estimating visual characteristics of 2D scatterplots via CNN

Computing individual Scagnostics scores for individual scatterplot is reasonably fast (in terms of milliseconds) [6]. However, it is still relatively slow for applications that need to do the Scagnostics computations extensively such as Outliagnostics [7] and MTSAD [8] for their real-time monitoring purposes. For these scenarios, the Scagnostics computation is still considered as having computationally expensive algorithms. Moreover, these algorithms are sensitive to the slight changes in the underlying data distribution within the scatterplot [9]. Concomitantly, with its recent advancements, Convolutional Neural Network (CNN) is gaining traction and has several state-of-the-art results in computer vision tasks. Therefore, ScagCNN [9] proposes to use CNN models to estimate the Scagnostics scores (prediction model) and classify set of scatterplots to typical Scagnostics types (classification model). ScagCNN aims to improve the Scagnostics computation time and reduce the sensitivity to the small shifts in the distribution of the underlying data. This section describes the architecture of ScagCNN, the configurations of its prediction and classification CNN models, and their accuracy.

5.1 ScagCNN architecture

Figure 11 depicts ScagCNN's schematic overview. Specifically, it uses real-world and synthesized data to train, validate, and test the CNN prediction and classification models. Using real-life datasets allows our solution to work in practice. In comparison, the synthesized one provides the ScagCNN with ground-truth labels while training classification models to detect typical Scagnostics types (the nine measures). In other words, these artificially generated plots have labels required to train the ScagCNN classification model. Also, ScagCNN uses ScagnosticsJS library [6] to compute the Scagnostics scores for these training, validating, and testing datasets.

The scatterplot data is converted into binary images and used as inputs for ScagCNN. Specifically, it also normalizes every variable's values into a unit range (i.e., [0, 1]). The normalized data then undergoes the binning step with 40×40 binning resolution (as suggested by the original Scagnostics paper). There are several reasons for doing this. First, normalizing and binning are the two data preprocessing steps done in the conventional approach. Please refer to Section 3 for why these steps are essential. Moreover, CNN models often require their inputs to have the same shape. Therefore, ScagCNN sets the binning resolution to 40×40, as mentioned in the original Scagnostics's paper. Notably, Scagnostics's computation algorithms do not take the number of data points within an individual bin into account (it is a bin if there is at least a point that falls within it and not otherwise, careless of the number of points inside each bin). Therefore, it's natural to view the

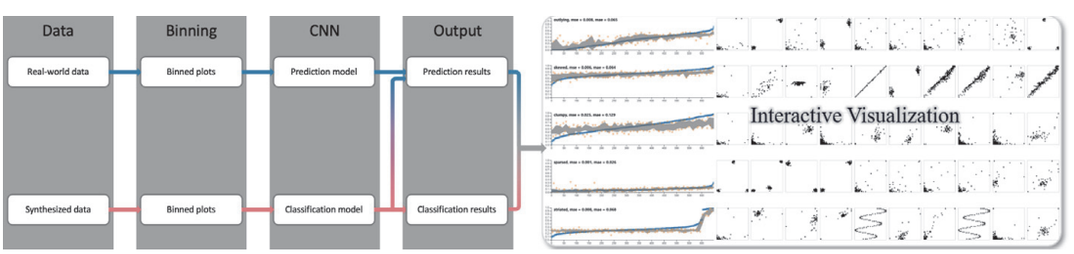

Figure 11.
ScagCNN [8] main stages: 1) data processing 2) binning, 3) building CNN prediction and classification models, 4) using the learned models to give results.

binned data as binary images when using them as inputs for the classification/prediction models.

ScagCNN models include a prediction model and a classification model. The former estimates the Scagnostics scores, while the latter classifies the underlying set of data points into a type that corresponds to one of the nine Scagnostics measurements. Both models use binary images, resulting from applying a binning method to the original data points as their inputs. The use of the synthesized dataset is necessary because it has the Scagnostics type as the ground-truth labels to train the classification model. The labels are the result of the generation process of this artificial dataset. Specifically, each generated set of data points is designed to have a high value for a specific Scagnostics measurement and is assigned to the corresponding Scagnostics type. Contrariwise, ScagCNN prediction model can leverage both real-world and synthesized data for its training, validation, and testing stages.

Artificial neural networks have high predictive power thanks to their freedom to learn abstract, salient features from the underlying data. This predictive power comes at the cost of explainability. In other words, the learned features are too abstract that humans cannot interpret their meanings. The lack of interpretability circumvents us from deploying our learned models into real-life since we cannot assure our models actually learn from appropriate features or merely memorize the underlying data's trivial characteristics. Consequently, there is a need to make them transparent [20]. ScagCNN accommodates the need for interpretability by offering users a web page. This web page visualizes the intermediate results of the algorithms involved in the Scagnostics computation. It also visualizes the features that the CNN models extracted in their various convolutional layers. These two tasks allow the users to debug and interpret the accuracy of the ScagCNN models. Specifically, the former will enable us to see if there is a bug in each of the underlying Scagnostics algorithms. Simultaneously, the latter allows the users to check if the extracted features are relevant or trivial. After having a thorough knowledge of these models' result generation process, users can make an educated decision regarding their accuracy.

5.2 ScagCNN prediction and classification models

Training an efficient CNN model involves experimenting with various architectures and tuning many hyperparameters. It is impossible to exhaustively search through many possible combinations of these configurations and hyperparameters to build a good CNN model. Consequently, a common approach in practice is to investigate and tune the existing CNN models appropriate for our specific domain (computer vision in this case). They are AlexNet [21], GoogleNet [22], and VGG-16 [23], to name but a few. These state-of-the-art models give excellent results in various computer vision tasks. However, in VGG-16, Visual Geometry Group (VGG) [24] provides an invaluable, reusable basic block for building CNN models following its standard [25], called a VGG block. Specifically, designers can stack one or more such blocks while building CNN models for their problem at hand. This block with guided architecture and hyperparameters helps reduce the number of possibilities that we need to search on while building our models.

Even using VGG as the building block, there are still many possibilities to experiment. Therefore, our approach starts with one VGG block and then stacks more VGG blocks and tests if more blocks result in improved validation accuracy. Suppose the training accuracy gets significantly higher than the validation one. In that case, we will start exploring techniques for overcoming overfitting, such as dropout, early-stopping, and weight regularization, and batch normalization. Once

the overfitting issue is solved, we can continue stacking more VGG blocks and check if going deeper helps increase the performance. Finally, in this specific case, experiments show that dropout and early stopping help tackling overfitting issues while batch normalization and weight regularization do not. **Table 1** summarizes the experimented number of VGG blocks and overfitting techniques used while training ScagCNN models. Going from left to right means stacking more VGG blocks, and "*Y*/*N*" means the corresponding layer is used or not correspondingly.

Configuration 1 is the simplest one, which uses a single VGG block and results in a testing MSE of 0.0155. Configuration 2 adds one VGG block, which helps to reduce the MSE to 0.0136. Similarly, configuration 3 adds another VGG block. However, it does more harm than good due to overfitting. Therefore, after every VGG block and the fully connected layer (FC1), a dropout layer is used to tackle the overfitting issue. The training histories show that the overfitting is not devastating. Thus, we used 0.1 as the dropout rate. There are also attempts to then train with 1, 2, and 3 VGG blocks combined with dropouts in configurations 4, 5, and 6, respectively. Notably, having dropouts helps to improve the performance (validation MSEs reduce significantly). However, adding more than two VGG blocks reduces the model's performance. Lastly, training neural networks with batches of data may help reduce training time and improve generalization. ScagCNN achieves the former advantage via parallelization. The latter is due to the weight updates for batches use the average of their gradients generated on a group of data items instead of a single one (in stochastic case). Configurations 1 to 6 use a batch size of 64, and configurations 8, 9, and 10 are experiments with batch-size set to 'None.'

Finally, ScagCNN uses configuration number 8 in **Table 1**. This configuration has two VGG blocks (VGG1 and VGG2), two fully connected layers (FC1 and FC2), and three dropout layers. There is one dropout layer after every layer except for the output layer (FC2). Both VGG1 and VGG2 have a convolution layer with 32 and 64 filters of size (3×3), respectively. Each of the VGG blocks also has a max-pooling layer with a pool size of (2×2) to reduce the size of the feature map generated by every VGG block. FC1 and FC2 have 128 and nine hidden units, respectively. The 128 hidden units of FC1 is to increase the non-linearity, thus improves the model's predictive power. The nine hidden units in FC2 correspond to the nine Scagnostics measurements. Notably, though not depicted in the table, all the convolutional fully connected layers are followed by a ReLU [26] activation function. The classification model also follows configuration 8 in **Table 1**. Because this architecture can extract

No.	VGG1	DO1	VGG2	DO2	VGG3	DO3	FC1	DO4	FC2	Batch	MSE[a]
1	Y	N	N	N	N	N	Y	N	Y	64	0.0156
2	Y	N	Y	N	N	N	Y	N	Y	64	**0.0136**
3	Y	N	Y	N	Y	N	Y	N	Y	64	0.0145
4	Y	Y	N	N	N	N	Y	Y	Y	64	**0.0121**
5	Y	Y	Y	Y	N	N	Y	Y	Y	64	0.0122
6	Y	Y	Y	Y	Y	Y	Y	Y	Y	64	0.0127
7	Y	Y	N	N	N	N	Y	Y	Y	None	0.0120
8[b]	Y	Y	Y	Y	N	N	Y	Y	Y	None	**0.0108**
9	Y	Y	Y	Y	Y	Y	Y	Y	Y	None	0.0114

[a] *Mean Squared Error.*
[b] *Selected configuration.*

Table 1.
Experimented CNN settings and their MSEs on the test dataset.

salient features useful for the Scagnostics score predictions and then it should also be appropriate for the Scagnostics classification task. However, for the classification task, we replace the activation function of the output layer (FC2) to softmax [26] instead of ReLU.

5.3 ScagCNN prediction and classification results

Table 2 summarizes the prediction results using the learned model to predict the Scagnostics scores on the testing dataset. "MSE" (mean-squared-error) and "MAE" (mean-absolute-error) are the two metrics that we use to measure the performance of the prediction model. Notably, the Scagnostics measurements have MAE less than or equal to 0.1, except clumpy. Scagnostics clumpy calculation depends on two edges in the runt statistics (one from the considering edge and another from the shorter sub-tree). Thus, this score is not robust to small changes in the underlying data [9].

Similarly, **Figure 12** depicts the sampled classification accuracy on synthesized and real datasets in panels (a) and (b), respectively. There are ground-truth labels for the synthesized dataset. Thus, it's straightforward to perform the accuracy evaluation on this set, and the accuracy is 98%. The high accuracy score may due to the generation bias of the synthetic data. Specifically, we generate the scatter plots of a type in such a way that it will flag high values for its corresponding Scagnostics measurement. Contrariwise, we do not have the ground-truth labels on the real dataset. Thus, we first sampled 100 scatterplots from the real-life test set, performed the classification on these plots, and asked two Scagnostics experts to evaluate the predictions' accuracy. The qualitative evaluation revealed that, out of 100 predicted labels, nine of them are incorrect. Notably, eight of them are related

	Outlying	Skew	Clumpy	Sparse	Striated	Convex	Skinny	Stringy	Monotonic
MSE	0.008	0.006	0.025	0.001	0.008	0.018	0.011	0.005	0.007
MAE	0.065	0.064	0.129	0.026	0.068	0.100	0.085	0.057	0.063

Table 2.
Mean Squared Error (MSE) and Mean Absolute Error (MAE) for 9 Scagnostics scores using conventional Scagnostics algorithms vs. ScagCNN predictions.

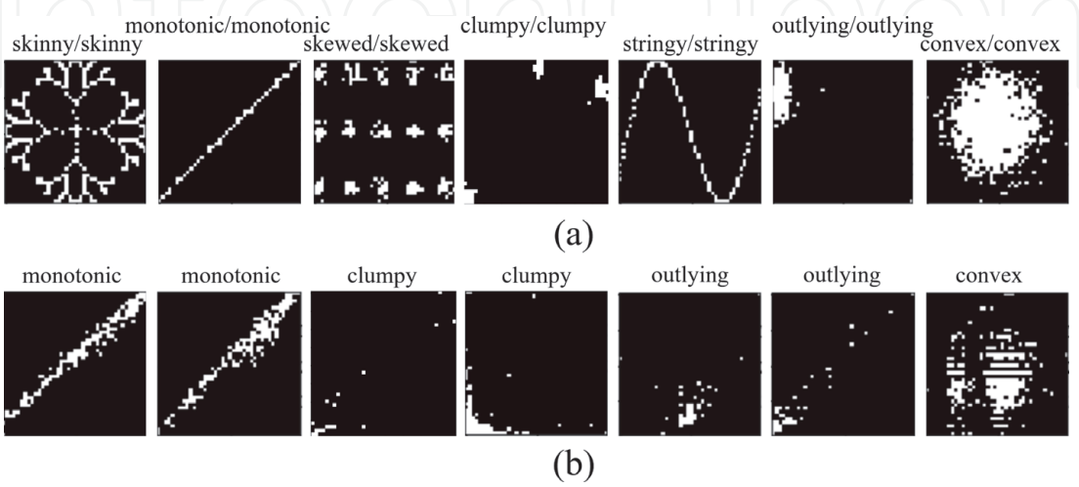

Figure 12.
Classification results for randomly selected scatterplots from synthesized and real datasets. Panel (a) depicts the classification results on the synthesized test set. The title is in the form of ground-truth/predicted *labels. Panel (b) shows the predicted labels for the real-life test data [9].*

Task	Time (ms)
Average scagnostics time per plot	20.019
Scagnostics time per plot standard deviation	20.354
Average model loading time	86.032
Average time for initial prediction	25.226
Average time per plot, using model	1.630
Time per plot standard deviation, using model	0.819

Table 3.
Machine learning model vs. Scagnostics algorithm runtime evaluation, evaluated on 589 scatterplots [7].

to the clumpy measurement. A large number of wrongly classified plots for clumpy scores suggest that we did not generate this kind of data properly, or the clumpy score is too sensitive [9].

5.4 ScagCNN run-time evaluation

There are two main advantages of using ScagCNN compared to the conventional Scagnostics algorithms. The first is that thanks to CNN's spatial invariant property, ScagCNN is robust to small changes in the underlying data. We refer interested readers to the original paper [9] for detailed analysis concerning Scagnostics's sensitivity and ScagCNN's robustness concerning small variants in the underlying data. This section discusses the second advantage of ScagCNN's. It reduces prediction time in case we need to predict the scores in batches.

Table 3 summarizes the time performance on the test dataset with 589 scatterplots from the real-life datasets. The testing device is an iMac with 3 GHz 6-Core Intel Core i5, macOS Catalina Version 10.15.3, 8 GB of RAM. However, one may access the project's Github page to perform the runtime evaluation on his/her device. Notably, ScagCNN model has enormously reduced the computational time (20.019 *ms* vs. 1.630 *ms*). Furthermore, the conventional Scagnostics computation time depends on the number of bins resulting from the binning process. Thus, the conventional approach has a large standard deviation regarding the computation times.

In contrast, all inputs into ScagCNN are images of the same size, so the computation times are relatively more stable. However, it takes time to load the ScagCNN model and perform the first prediction. This initialization time is due to the WebGL shader compilation needed for the model [26]. Therefore, we should always use ScagCNN in batch to gain time performance. Further investigations reveal that ScagCNN prediction model achieves better performance by leveraging parallelisms (multi-cores) and using more memory space [7].

Finally, ScagCNN provides a web prototype for readers to evaluate the prediction results qualitatively. The source codes and result visualizations are available on the project's web page: https://idatavisualizationlab.github.io/V/ScagCNN.

6. Conclusions

This chapter introduces a recent Scagnostics implementation called ScagnosticsJS. Its implementation in JavaScript fosters the use of visual features on the web platform. Also, its extensions of Scagnostics to 3D and nD versions promote visual features that characterize the rapidly growing multivariate data. This chapter

also describes Outliagnostics as an attempt to demonstrate the use of ScagnosticsJS 2D version on the web. This application uses a leave-one-out strategy and Scagnostics outlying score to detect individual data entry's outlying contribution to the overall set of points over time. Likewise, MTSAD is another application to illustrate the use of ScagnosticsJS nD version in finding the abnormalities in multivariate time series data. Finally, this chapter discusses a recent attempt to use convolutional neural networks to predict Scagnostics scores and classify scatterplots into their typical Scagnostics types. Its main purposes are to tackle the issues that the conventional Scagnostics algorithms involve time-consuming algorithms, and they are also sensitive to slight changes in the underlying data.

Acknowledgements

This research is supported in part by the National Science Foundation under the I-Corps award number 2017018, grant CNS-1362134, OAC-1835892, and the IUCRC-CAC (Cloud and Autonomic Computing) Dell Inc. membership contribution.

Author details

Vung Pham* and Tommy Dang
Texas Tech University, Lubbock, USA

*Address all correspondence to: vung.pham@ttu.edu

References

[1] Wilkinson L, Anand A, Grossman R. Graph-theoretic scagnostics. Proceedings - IEEE Symposium on Information Visualization, INFO VIS. 2005:157–164.

[2] Lee W, Anushka A. Compute scagnostics - scatterplot diagnostics. rforge; 2018. Available from: https://www.rforge.net/scagnostics/.

[3] Josua K. Python binding to R scagnostics.. github; 2015. Available from: https: //github.com/nyuvis/scagnostics.

[4] Dang TN, Wilkinson L. ScagExplorer: Exploring scatterplots by their scagnostics. IEEE Pacific Visualization Symposium. 2014:73–80.

[5] Fu L. Implementation of Three-dimensional Scagnostics [master's thesis]. University of Waterloo; 2009.

[6] Pham V, Dang T. ScagnosticsJS: Extended Scatterplot Visual Features for the Web. In: Wilkie A, Banterle F, editors. Eurographics 2020 - Short Papers. The Eurographics Association; 2020..

[7] Pham V, Dang T. Outliagnostics: Visualizing Temporal Discrepancy in Outlying Signatures of Data Entries. In: 2019 IEEE Visualization in Data Science (VDS). Vancouver, BC, Canada, Canada: IEEE; 2019. p. 29–37.

[8] Pham V, Nguyen N, Li J, Hass J, Chen Y, Dang T. MTSAD: Multivariate Time Series Abnormality Detection and Visualization. In: 2019 IEEE International Conference on Big Data (Big Data). IEEE; 2019. p. 3267–3276.

[9] Pham V, Nguyen NV, Dang T. ScagCNN: Estimating Visual Characterizations of 2D Scatterplots via Convolution Neural Network. In: Proceedings of the 11th International Conference on Advances in Information Technology; 2020. p. 1–9.

[10] Wilkinson L, Wills G. Scagnostics distributions. Journal of Computational and Graphical Statistics. 2008;17(2): 473–491.

[11] Bostock M, Ogievetsky V, Heer J. D3 data-driven documents. IEEE transactions on visualization and computer graphics. 2011;17(12):2301–2309.

[12] Nürnberg R. Calculating the area and centroid of a polygon in 2d. URL: http://wwwf imperial ac uk/˜rn/centroid pdf. 2013.

[13] Dang TN, Wilkinson L. Transforming scagnostics to reveal hidden features. IEEE Transactions on Visualization and Computer Graphics. 2014;20(12):1624–1632.

[14] Sartipizadeh H, Vincent TL. Computing the approximate convex hull in high dimensions. arXiv preprint arXiv:160304422. 2016.

[15] WorldWebMath. N Dimensional Geometry; 1997. Last accessed 11 June 2019. Available from: http://web.mit. edu/wwmath/vectorc/ndim.html.

[16] Asuncion A, Newman DJ. UCI Machine Learning Repository. University of California; 2007. Available from: http://www.ics.uci. edu/˜mlearn/MLRepository.html.

[17] Dang T. Visualizing Multidimensional Health Status of Data Centers. In: Bhatele A, Boehme D, Levine JA, Malony AD, Schulz M, editors. Programming and Performance Visualization Tools. Cham: Springer International Publishing; 2019. p. 273–283.

[18] Pham V, Nguyen N, Dang T. ContiMap: ContiMap: Continuous Heatmap for Large Time Series Data. In:

2020 IEEE Visualization in Data Science (VDS). Virtual Conference: IEEE; 2020.

[19] Hartigan JA. Clustering Algorithms. New York: John Wiley & Sons; 1975.

[20] Le DD, Pham V, Nguyen HN, Dang T. Visualization and Explainable Machine Learning for Efficient Manufacturing and System Operations. Smart and Sustainable Manufacturing Systems. 2019 Feb;3(2):20190029. Available from: https://doi.org/10.1520/ssms20190029.

[21] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems; 2012. p. 1097–1105.

[22] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2015. p. 1–9.

[23] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:14091556. 2014.

[24] Visual Geometry Group. Visual Geometry Group. http://www.robots.ox.ac.uk/~vgg/;. Accessed: 2019-11-4. http://www.robots.ox.ac.uk/~vgg/.

[25] Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning. https://d2l.ai; 2019. Accessed: 2019-11-4.

[26] TensorFlow. Tensorflow documentation. TensorFlow;. Accessed: 2019-12-24. https://github.com/tensorflow/tfjs/tree/master/tfjs-converter.