ScagCNN: Estimating Visual Characterizations of 2D Scatterplots via Convolution Neural Network

Vung Pham
Texas Tech University
Lubbock, Texas, US
vung.pham@ttu.edu

Ngan V.T. Nguyen
Texas Tech University
Lubbock, Texas, US
ngan.v.t.nguyen@ttu.edu

Tommy Dang
Texas Tech University
Lubbock, Texas, US
tommy.dang@ttu.edu

ABSTRACT

Scagnostics is a set of visual features that characterizes the data distribution of a 2D scatterplot and has been used in a wide range of applications. However, calculating the scagnostics scores involves computationally expensive algorithms. Moreover, the algorithms are sensitive to the slight changes in the underlying data distribution within the scatterplot. Therefore, this work provides a machine learning model, called ScagCNN, to estimate the scagnostics scores. This model aims to improve the scagnostics computation time and to reduce the sensitivity to the small shifts in the data distribution. This work also provides a web prototype to explore the predictive performance of the model and to give a visual explanation about whether a prediction is accurate. Furthermore, we test the performance of our solution on datasets of various sizes.

CCS CONCEPTS

• **Human-centered computing → Visual analytics**.

KEYWORDS

Visual features, Scagnostics, Convolution Neural Network

ACM Reference Format:
Vung Pham, Ngan V.T. Nguyen, and Tommy Dang. 2020. ScagCNN: Estimating Visual Characterizations of 2D Scatterplots via Convolution Neural Network. In *The 11th International Conference on Advances in Information Technology (IAIT2020), July 1–3, 2020, Bangkok, Thailand*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3406601.3406644

1 INTRODUCTION

Visual analytics concerns with coupling interactive visual representations and underlying analytical processes (e.g., statistical procedures, feature extraction techniques) so that users can utilize their cognitive and reasoning capabilities to perform complex tasks or to make decisions effectively. An example of such visual analytical reasoning is the Scagnostics [30], which aims to capture the shape, density, correlations, and texture of data point distributions in a 2D space. The problem of visual characterization is that it is computationally expensive. In this paper, we use the Neural Network to learn these visual features and predict the numerical outputs without computing the expensive proximity graphs, which are the backbones of the computation processes of these features. This time improvement is essential for real-time visual analytics on a large amount of streaming data such as monitoring of the critical variables from high performance computing center [19, 21].

In particular, we use Convolution Neural Network (CNN) for learning and estimating the visual features of scatterplots without calculating the underlying proximity graphs that are all subsets of the Delaunay triangulation: the minimum spanning tree (MST), the alpha complex [10], and the convex hull. The approach can be naturally expanded to other visual analytics applications, such as time series, parallel coordinate, or pixel-based representations. Figure 1 depicts the main ideas of ScagCNN framework and the details are given in Section 3. Thus, the contributions of this paper are three-fold:

- We propose a novel approach for visual analytics computation based on machine learning. We argue that visual analytics methods are more computationally expensive, while the machine learning approach (based on visual analytics) can save computational time and is more stable as it conveys the knowledge from the crowd. The model is trained once and can be used for testing without building the backbone algorithms for each of the new instances.
- We develop an interactive interface exploring the visual features and make sense of the learning process using CNN. We also allow users to investigate and compare the results of visual analytics and machine learning models side by side.
- We demonstrate our approach on various real-world and synthetic data sets. Although demonstrated on scatterplot visual features, our proposed approach has general implications: it can be easily extended to other visual analytics models, such as time series, parallel coordinates, and matrix visualizations.

This paper is organized as follows: The next Section presents related research in visual features and popular techniques for visual analytics vs. machine learning. Section 3 presents our visual characterization and provides the architecture of our ScagCNN interactive prototype for exploring the learning output compared to the visual features generated by the original Scagnostics algorithms. Section 4 discusses the CNN models built for classification of scagnostics typical scatterplots and for predicting scagnostics scores. Use cases and performance evaluations of ScagCNN are demonstrated in Section 5. Lastly, Section 6 concludes the paper and presents future direction for this work.

IAIT2020, July 1–3, 2020, Bangkok, Thailand

ACM ISBN 978-1-4503-7759-1/20/07…$15.00
https://doi.org/10.1145/3406601.3406644

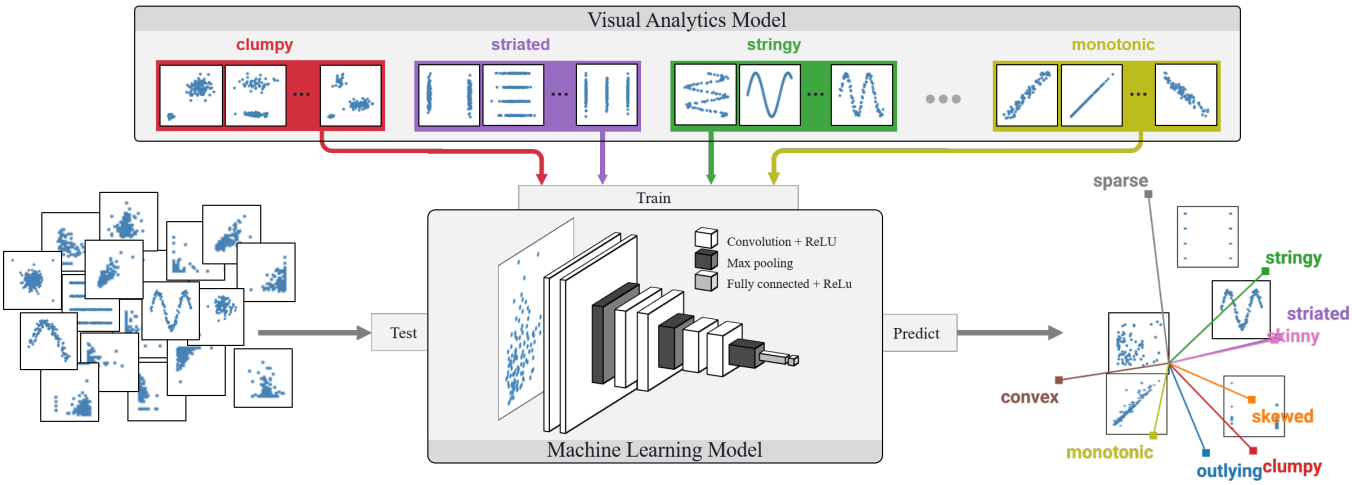


Figure 1: Overview of our ScagCNN framework: The CNN models are trained on the visual characterizations generated by the Visual Analytics component (top). The trained CNN models are then used to classify or estimate the visual scores of new plots without involving the Visual Analytics algorithms.

2 RELATED WORK

As the purpose of this paper, this section only surveys the works that apply machine learning in visual quality predictions. For interested readers, a complete review of recent researches in visual quality metrics [6] can be found in [4]. Recent advancements in the field of machine learning give rise to the number of works in applying machine learning methods to learning from the human-labeled visual quality and then use the learned model to predict the quality of the unseen visualizations. For instance, Aupetit and Sedlmair [3, 24] evaluated state-of-the-art class separation measures using human judgments on the class separation of color-coded 2D scatterplots. ScatterNet [17] uses a set of CNN [16] layers to transform images of scatterplots into a set of latent features. These learned features are then used to compare the similarities among scatterplots. This work is highly related to ours as we both leverage the CNN ability to learn features from images of scatterplots. However, ScatterNet aims to learn visual quality from the human-labeled scatterplot images. In contrast, we target the predictions of scagnostics scores for scatterplots using the CNN model to improve the time and robustness.

CNN has current-state-of-the-art results in various computer vision tasks such as handwriting recognition [16] and faces recognition [1]. It achieves these good results thanks to its ability to extract features regarding the spatial properties of the data. There are various architectures and hyperparameters to be optimized to produce an efficient CNN model. It is nearly impossible to try all the possibilities extensively. Therefore, we investigated the state-of-the-art CNN models in solving different computer vision tasks such as AlexNet [14], GoogleNet [26], and VGG-16 [25]. While all three give good results, VGG-16 provides a general template to guide subsequent researchers in designing new neural networks [34] while solving different computer vision tasks. One such template to produce an efficient CNN model is from the Visual Geometry Group (VGG) [28] called VGG block.

Regarding sensitivity analysis [23] of Scagnostics, Wang et al. [29] experimented with both artificial settings and user studies. They then reported that Outlying and Clumpy scores are sensitive to binning, and humans are not sensitive to small changes in the underlying data, but scagnostics algorithm is. They proposed to use (1) adaptive binning, (2) a hierarchy-based scagnostics called *RScag*. Adaptive binning combines the original hexagon binning [19] and randomly sample n points out of m points in each hexagon cell ($n = \gamma \times m$, where γ is a predefined ratio) to preserve the densities of the points [5]. To improve the robustness of the Outlying score, they divide the considering scatterplot into clusters and identify the outlying edge length threshold (ω) per cluster instead of calculating this threshold globally. To tackle Clumpy sensitivity, they proposed to incorporate the longest edges in both sub-clusters and also consider the number of points in each of them while calculating the Clumpy score.

Though these strategies help in improving the robustness of scagnostics, we argue that these approaches have several issues. There are two obvious issues with adaptive binning. First, it takes a longer time. Second, it involves random sampling leading to non-deterministic behaviors (i.e., different executions may give different scagnostics scores due to the randomness). Another better approach to binning, which is faster, deterministic, and can reserve the original density is to use leader binning [19, 20]. Hierarchy-based scagnostics also has limitations. First, it takes time to find the clusters (i.e., it slows down the scagnostics score computations). Second, in case there are no clear clusters (demonstrated in our findings in Section 5.3), this approach does not work. In this paper, we leverage machine learning models to predict scagnostics measurements to improve the computation time and robustness of the scagnostics scores [31].

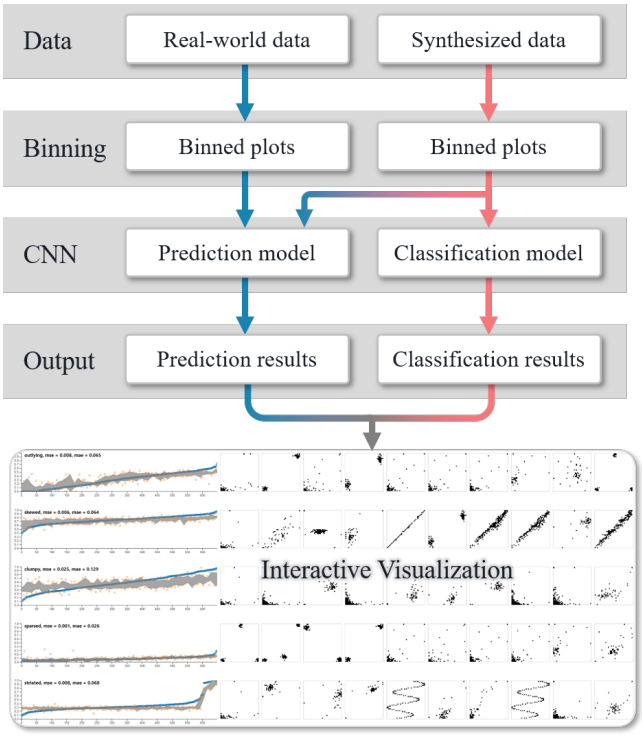


Figure 2: Major steps in our ScagCNN framework.

3 SCAGCNN SCHEMATIC OVERVIEW

Figure 2 depicts the schematic overview of our framework. The datasets used to train, validate, and test the machine learning models in ScagCNN are both real-life and synthesized. The real-life datasets are to assure that our solution works in practice. On the other hand, the synthesized data enables our ScagCNN model to have data to detect typical types of scatterplots corresponding to the nine scagnostics measures. A scatter plot is said to be typical for a specific scagnostics measure type if it has a high score for that measurement, and at the same time, has as low as possible for the others. These artificially generated plots also provide us the labels required to train our classification model to classify a given 2D scatterplot into one of the nine scagnostics measurement types. *ScagnosticsJS* library [20] is used to compute the scagnostics scores for training and testing the prediction models. Section 4.1 and Section 4.2 detail the steps to process and generate real-life and synthesized datasets and scagnostics score computations used in this work.

Similar to the original scagnostics computation process, we normalize each variable of the data into [0, 1] range and bin the input data with the binning resolution of 40×40. These steps are to ensure that the data fed into training and testing CNN models are similar to those fed into conventional scagnostics algorithms. The normalization of each variable in the data into unit range is to make sure that one variable does not dominate the other due to different value ranges. Furthermore, it is crucial to have the same image resolutions to train/test CNN models. Also, algorithms in scagnostics computation process do not take the density of an individual bin (number of data points inside a given bin) into account. They only consider whether there is a data point in a bin. For that reason, the binning algorithm used in generating ScagCNN data converts scatterplot data into binary values (i.e., 1s for the cells which as data and 0s otherwise). The binned binary image helps to reduce the data and the corresponding model size, thus reduces training time [9].

The generated binary images are then used to train/test the classification and prediction models in ScagCNN. The first is to classify a given input scatterplot into one of the nine scagnostics measurement types. On the other hand, the latter is to predict the nine scagnostics scores for a given input scatterplot. We use the synthesized dataset to train and test the classification model. The reason is that the training process of the classification model requires labels for the data instances. The labels of the synthesized scatterplots are available as the types of the scagnostics measures that they are generated for. On the other hand, we use both the real-life datasets and the synthesized one to train and test the scagnostics score prediction model. Also, we run the scagnostics algorithm to compute the scagnostics measures for each of the scatterplots to have the actual scagnostics scores for training the prediction model. Sections 4.4 and 4.3 detail the architectures of the classification and prediction models in ScagCNN correspondingly.

The learned CNN models are black-box. They need to be transparent to users so that they can understand the rationale behind the outcomes and possibly modify the algorithms/models to improve the accuracy [15]. Therefore, ScagCNN provides users an interactive web prototype to visualize the outputs from the models. The interactive visualization prototype aims to explain the intermediate results of the algorithms in the underlying scagnostics computation process. It also allows the user to see the extracted features that the CNN models learned in its intermediate stages. These two are especially helpful in the case that the scagnostics computed scores and the predicted scores from the learned CNN model deviate significantly. The explanations allow the users to have a thorough understanding of the result generation process before deciding which result is more appropriate. Section 5 details the ScagCNN visualizations and interactions.

4 CNN MODELS

4.1 Datasets

The datasets used for training and testing ScagCNN framework are both real-world and synthesized. For the synthesized dataset, there are 100 scatterplots generated for each scagnostics measure. These 100 scatterplots of each typical scagnostics measurement type have high scores for their corresponding type. On the other hand, for all the real-life datasets, as an effort to add noises to the datasets and generate more data, we augmented the real-life datasets by adding random noises, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, to every variable value in the scatterplots. Where $\sigma = 1/40$ is selected as the bin-size as the effort not to distort the original data too much. Specifically, a dataset with n original scatterplots is augmented ten times leading to a total of $n + n \times 10$ scatterplots in our resulted datasets. Table 1 gives information about these two datasets.

The first two datasets are from the Bureau of Labor Statistics (BLS) [8]. The US Unemployment Rate dataset contains the men

and women yearly unemployment rate of 51 States in 19 years from 1999 to 2017. The US Employment Net Change dataset contains all employees (in thousands, Month Net Change, seasonally adjusted) in Good Producing and Service Providing industries of 53 States from January 2000 to August 2018.

The next three datasets were retrieved from Kaggle[11]. The International Debt dataset from World Bank Open Data Repository [33] contains information for 124 countries from 1970 until 2024 (with projections), in particular, the debt and the population features of this dataset are used in our application. World Terrorism dataset is from National Consortium for the Study of Terrorism and Responses to Terrorism (START) [18] contains terrorism data for 205 countries over 47 years (from 1971 to 2017), and we make use of the number attacks versus the number of killed variables in our use-case. The New York Stock Exchange dataset contains the information of 501 listed Stocks from January 2011 until December 2017, the price and volume dimensions of this dataset are used in our application.

The next datasets are the World Bank Open Data Repository retrieved from UIC repository [2]. The Prevalence Of HIV dataset contains information about the prevalence of HIV female (ages 15-24) and male (in the same age range). The World Unemployment Rate dataset contains records about female and male percentage of the labor force for 241 countries over 56 years. The World Life Expectancy dataset contains male and female life expectancy of 263 countries all over the world in 26 years. To add to the variety of our datasets, we also collect health status from a university High-Performance Computing Center. This dataset contains fan speed and CPU temperature measurements of 467 CPUs at 52 time-steps.

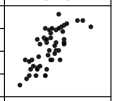
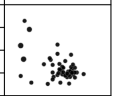
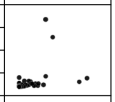
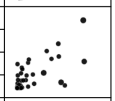
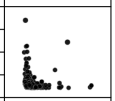
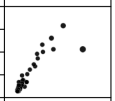
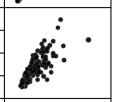
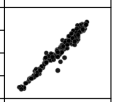
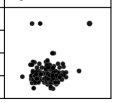
Both types of datasets add up to 7,369 scatterplots used to train and evaluate our models. One-tenth of the dataset (737 scatterplots) is used for testing purposes, and the rest is used for training. Also, the training set is further divided into training and validation sets with 2/3 and 1/3 ratios correspondingly (train on 4,443 samples and validate on 2,189 samples). The validation set is used as the early stopping condition to choose the best model according to this validation data. This early stopping condition means that the training process stops if the performance does not improve after a certain number of training epochs (the patience parameter) and returns the learned model with the best performance on the validation set.

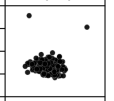
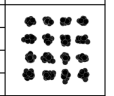
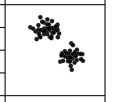
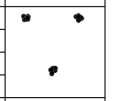
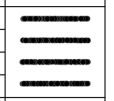
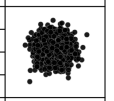
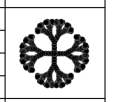
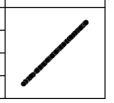
4.2 Main stages in scagnostics computation

We use *ScagnosticsJS* [20], a scagnostics implementation library for JavaScript, to calculate scagnostics scores for the scatterplots from the mentioned datasets. The scagnostics scores computation implementation includes several intermediate steps described in Figure 3.

(1) **Data Normalization** standardizes value ranges of different dimensions into unit range [0,1].
(2) **Binning** helps scagnostics calculation relatively independent of the total number of data points and therefore more scalable.
(3) **Triangulation**: scagnostics measures are calculated based on the subset of the Delaunay triangulation on the binned data, including MST, concave hull, and convex hull.

Table 1: Datasets and their exemplar scatterplots for training and evaluating ScagCNN.

Real-world Datasets		Example plot
US Unemployment Rate (USUER)		
Variable 1	Men	
Variable 2	Women	
# plots	19 + 19x10 = 209	
US Employment Net Change (USENC)		
Variable 1	Goods	
Variable 2	Services	
# plots	24 + 224x10 = 2,464	
International Debt Data (WBID)		
Variable 1	Debt	
Variable 2	Population	
# plots	55 + 55x10 = 605	
World Terrorism (WTRSM)		
Variable 1	Attacks	
Variable 2	Killed	
# plots	47 + 47x10 = 517	
New York Stock Exchange (NYSE)		
Variable 1	Price	
Variable 2	Volume	
# plots	84 + 84x10 = 924	
Prevalence Of HIV (WBHIV)		
Variable 1	Female	
Variable 2	Male	
# plots	26 + 26x10 = 286	
World Unemployment Rate (WUER)		
Variable 1	Female	
Variable 2	Male	
# plots	26 + 26x10 = 286	
World Life Expectancy (WBLE)		
Variable 1	Female	
Variable 2	Male	
# plots	56 + 56x10 = 616	
High-Performance Computing (HPCC)		
Variable 1	Temperature	
Variable 2	Fan speed	
# plots	52 + 52x10 = 572	

Synthesized Datasets		Example plot
Outlying		
Variable 1	x	
Variable 2	y	
# plots	100	
Skewed		
Variable 1	x	
Variable 2	y	
# plots	100	
Clumpy		
Variable 1	x	
Variable 2	y	
# plots	100	
Sparsed		
Variable 1	x	
Variable 2	y	
# plots	100	
Striated		
Variable 1	x	
Variable 2	y	
# plots	100	
Convex		
Variable 1	x	
Variable 2	y	
# plots	100	
Skinny		
Variable 1	x	
Variable 2	y	
# plots	100	
Stringy		
Variable 1	x	
Variable 2	y	
# plots	100	
Monotonic		
Variable 1	x	
Variable 2	y	
# plots	100	

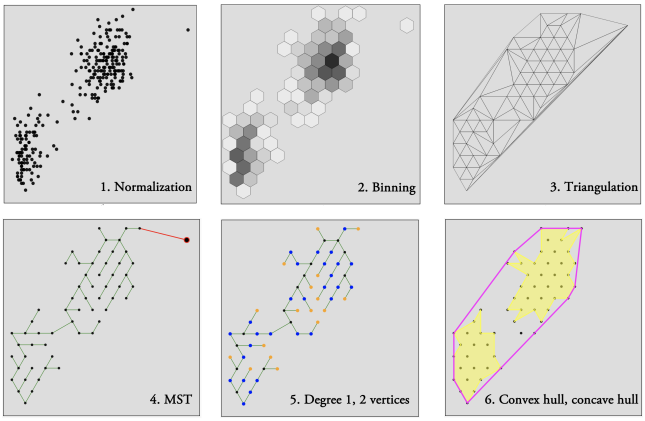


Figure 3: Intermediate stages in computing 2D scagnostics: (1) Normalization, (2) Binning, (3) Triangulation, (4) MST (with an outlier in red), (5) Degree 1 (orange) vs. Degree 2 (blue) vertices, and (6) Convex vs. Concave hull.

(4) **MST**: Six out of nine scagnostics measures (except Convex, Skinny, and Monotonic) are calculated based on the MST.

(5) **Degree 1, 2 vertices** are used to calculate the Striated and Stringy measures.
(6) **Convex hull vs. concave hull**: The Convex and Skinny are measured based on the perimeters and/or areas of these shapes.

4.3 Prediction Model

As discussed in Section 2, we use VGG block as the fundamental component of our ScagCNN models. Using the VGG block is selected as one of the efforts to reduce the need to try many different architectures and hyperparameters. However, it is still impossible to try all the configurations. Therefore, our strategy is to start with a simple neural network of one VGG block and increase the number of blocks until observing that going deeper (i.e., adding more VGG blocks) leads to overfitting. Overfitting means the training loss, e.g., mean squared errors (MSEs), continues to decrease, but that of the validation data does not decrease or even increases. Overfitting does not necessarily mean that proceeding further is bad since several techniques can be applied to overcome overfitting, and at the same time, make use of the useful features learned at the deeper layers.

Common techniques to tackle overfitting are dropout, weight regularization, batch normalization, and early stopping, to name but a few. Therefore, we also applied these techniques with different hyperparameters in our process to find CNN models for our solution. Via experiments, in this case, only dropout and early stopping help in reducing the validating errors while batch normalization and weight regularization do not. Table 2 summarizes the experimented configurations and their corresponding testing MSEs. The columns are ordered as the stacking order in the model architecture (left to right means going deeper). Also, "Y" means that the layer is used and "N" otherwise.

As of our experiments, we decided to use a CNN model with two VGG blocks, one Dense layer (Dense1), and a Dropout layer (with the dropout rate of 0.1) after each of the layers. Each of the VGG blocks (VGG1 and VGG2) contains two 2D convolution layers; both have (3, 3) filter size. VGG1 has 32 filters, and the number of filters is doubled in the next VGG block (i.e., there are 64 filters for VGG2). Each of them also includes a max-pooling layer with the patch size of (2, 2) to down-sample the extracted feature map into half every VGG block. Dense1 has 128 hidden units, and there is one additional fully connected layer (Dense2) with nine hidden units corresponding to the nine scagnostics measurement types to be predicted. All these layers use ReLU [27] as their activation function.

We started with the most straightforward architecture, which uses one VGG block only as in configuration 1 then add one more VGG block in configuration 2, which reduces the testing MSE from 0.0155 to 0.0136 correspondingly. It seems beneficial to go deeper. Therefore, we add one more VGG block in configuration 3. However, it does more harm than good. Figure 4 training histories for these three configurations (1, 2, and 3) on the left column. All these three training histories imply that overfitting occurred in these fitting processes of these models. Specifically, after a few training epochs, the training loss continues to decrease while the validation loss does not. Therefore, we decided to add a dropout layer after every VGG block and one after the Dense1 layer to tackle the overfitting issue. As observed from the training histories, the overfitting is not very serious; therefore, the selected dropout rate is 0.1. We then train the configuration 4, 5, and 6 with 1, 2, and 3 VGG blocks correspondingly. It is observable that adding dropout layers reduces the MSEs significantly. However, going deeper does not help in this case.

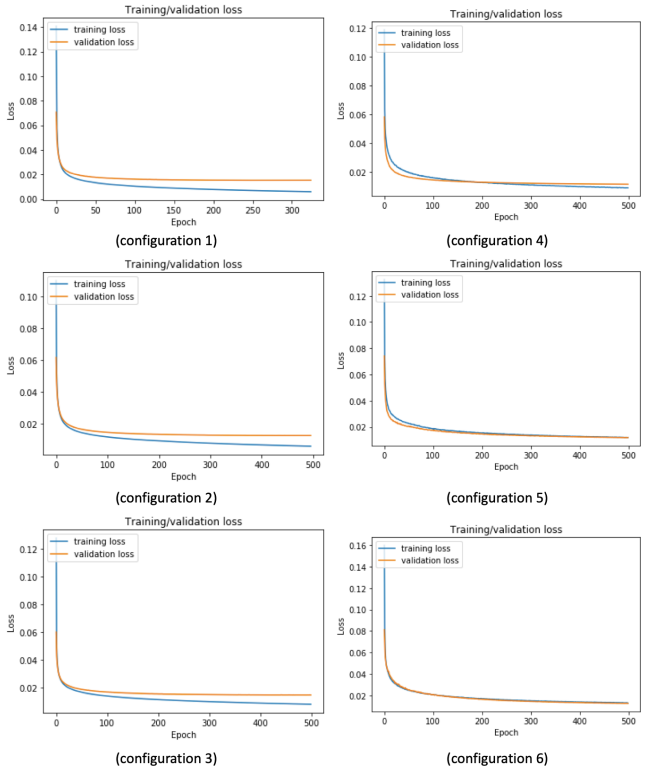


Figure 4: Training/validation losses over number of training epochs for configurations 1 to 6 (described in Table 2).

Using batch size during fitting helps to speed up the training time, and in some cases, allows the training process to be generalized. The batch-size generalization is achieved by only updating the weights of the model after fitting a batch of training instances instead of updating after every individual instance. However, it is observable from training histories of configurations 5 and 6 that the training/testing losses would continue to decrease if we increase the training epochs. Instead, in this case, we decided to test the training process without specifying the batch size and use the default batch size is one. Configuration 8, 9, and 10 are our experiments in this direction. It is clear that configuration 8, with 2 VGG blocks, a dropout layer with the dropout rate of 0.1 after every VGG block and the first dense layer, and default batch size, achieves the best performance out of the nine experimented configurations (approximately 0.0108 for the MSE).

4.4 Classification Model

Out of 900 scatterplots from the synthesized dataset, 90 of them are reserved for testing purposes. Other 810 scatterplots are divided

Table 2: Different experimented CNN configurations and corresponding MSEs on the test dataset.

No.	VGG1	DO1	VGG2	DO2	VGG3	DO3	Dense1	DO4	Dense2	Batch Size	Test MSE
1	Y	N	N	N	N	N	Y	N	Y	64	0.015565132
2	Y	N	Y	N	N	N	Y	N	Y	64	**0.013627671**
3	Y	N	Y	N	Y	N	Y	N	Y	64	0.014486085
4	Y	Y	N	N	N	N	Y	Y	Y	64	**0.012099808**
5	Y	Y	Y	Y	N	N	Y	Y	Y	64	0.012238170
6	Y	Y	Y	Y	Y	Y	Y	Y	Y	64	0.012689635
7	Y	Y	N	N	N	N	Y	Y	Y	None	0.012038863
8	Y	Y	Y	Y	N	N	Y	Y	Y	None	**0.010816987**
9	Y	Y	Y	Y	Y	Y	Y	Y	Y	None	0.011397023

with a ratio of 2/1 into training and validation sets correspondingly. After experimenting with different configurations for training the classification models, we also follow configuration 8 in Table 2 as the architecture for developing classification model with the assumption that such configuration can extract all useful spatial features from the input data. The only change to the architecture is to use the softmax activation [27] function at the last layer (Dense2) instead of ReLu in the prediction model case. Figure 5 shows the training/validation losses over the number of training epochs for the classification model.

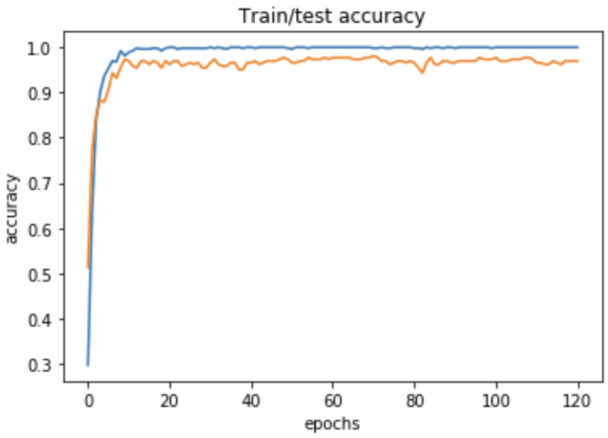


Figure 5: Training/validation losses over the number of training epochs for the classification model.

5 RESULTS AND VISUALIZATIONS

5.1 Prediction results

We provide a web prototype for readers to qualitatively evaluate the prediction results. Figure 6 depicts the summary of the results on the nine scagnostics measures for the 737 testing scatterplots. The labels "*mse*" and "*mae*" mean mean-squared-error and mean-absolute-error. Also, the y-axis represents the score, while the x-axis contains the plot IDs. Finally, the blue dots are the scores calculated using the scagnostics algorithm. It is worth noting that we do not display all the predicted scores since this leads to cluttering issues due to a large number of testing scatterplots. Instead, we use gray stream-graphs to visualize clusters of predicted points from the first to the third quartile and the orange dots for the other points. These orange dots are often plots with high errors. Thus, they help the analyzers to focus on while investigating the prediction results.

It is observable that all of the scores have MAE (mean absolute error) ≤ 0.1, except clumpy. The reason is that scagnostics clumpy measurement is sensitive to small changes while our CNN model is not. That is, scagnostics clumpy is sensitive to small changes because its calculation is determined by two edges (one from the considering edge and another one from shorter sub-tree [32]), which might not accurately characterize patterns within complex distributions [29]. Interested readers can refer to the web page of our project with interactive visualizations provided to investigate the prediction results further.

5.2 Classification results

Figure 7 shows the classification results on both synthesized (a) and real (b) data sets. In the case of synthesized testing data, we have class labels to validate, and the accuracy is 98%. This high accuracy value is reasonable since the synthesized scatterplots are designed to flag for their designated visual characterizations. On the other hand, for the real-life testing data, since there are no labels for the plots, we qualitatively evaluated the accuracy by sampling 100 scatterplots from the set of real-life data and used the model to assign a class label for each of the plots. We then let two experts in scagnostics manually evaluate the predicted labels. Interested readers can refer to the web page of our project for the details of this qualitative evaluation. Nine out of 100 predicted labels are incorrect. Furthermore, eight of the nine wrongly classified plots are related to Clumpy measurement. These may imply that our training data is not suitable, or the Clumpy score is too sensitive, as discussed in Section 5.3.

5.3 Robustness evaluation

We collected/generated the scatterplots with slight changes in the underlying data to test the robustness of our solution. The first strategy is to collect real time-series data with smooth transitions from one step to another. The second approach is to randomly delete/add one or more points to existing data. The experimental results imply that the machine learning approach is less sensitive to slight perturbations than the original scagnostics algorithm. For instance, Table 3 shows the Outlying scores for two scatterplots in the Prevalence of HIV dataset [2] (years 2011 vs. 2014) using scagnostics algorithm and machine learning models.

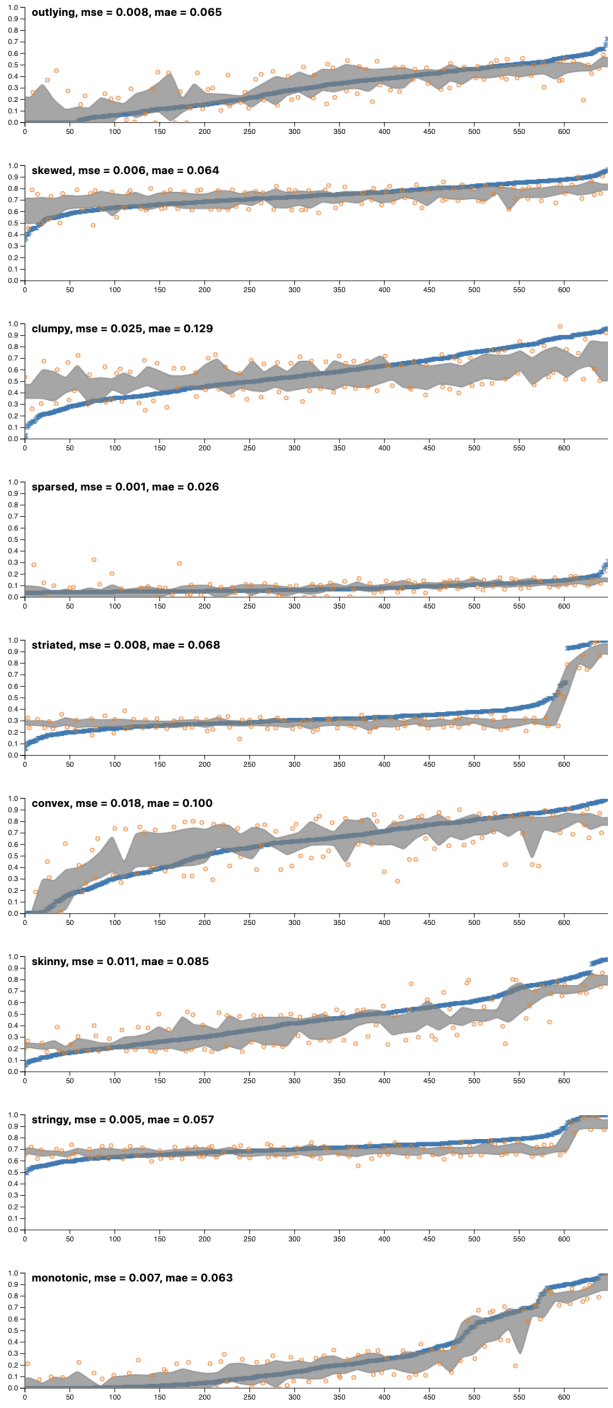


Figure 6: Prediction results: "*mse*" and "*mae*" are mean-squared-error and mean-absolute-error; y-axis represents the score; x-axis contains plot IDs; blue dots are scores calculated using scagnostics algorithm; gray stream-graphs are clusters of $1^{st} - 3^{rd}$ quartiles of the predicted scores to avoid cluttering; orange dots are predicted scores with high errors.

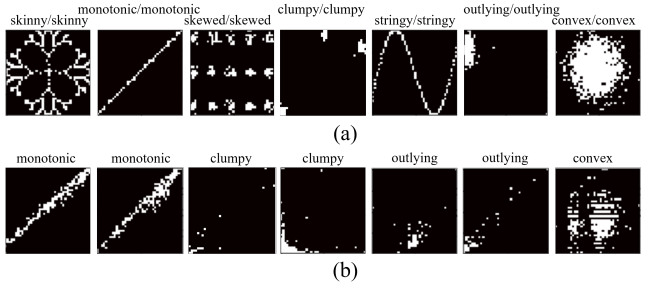


Figure 7: Classification results on synthesized (a) vs. real (b) data for some randomly selected scatterplots. On the top panel (a), these scatterplots are synthesized, and *label1/label2* means the actual and predicted labels correspondingly. On the bottom panel (b), these are scatterplots from real data sets, so the labels are the predicted ones.

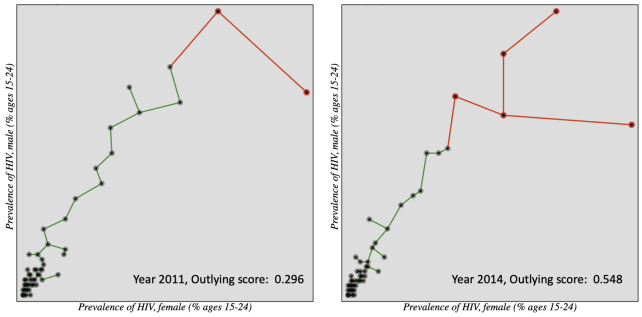


Figure 8: Slight modifications in the underlying scatterplots lead to changes in the MST structure (thus, the abrupt change in the outlying score from 0.296 to 0.5).

It is observable that with slight changes in the underlying data, the scagnostics algorithm changes abruptly from 0.296 outlying score to 0.548, while the machine learning models only changes slightly from 0.282 to 0.292. The sensitivity of the underlying scagnostics algorithms is due to the change in the MST structure even with slight changes in the bins, as shown in Figure 8. It is also worth noting that the hierarchy-based approach, as in RScag [29], cannot tackle this issue since the outliers are external outliers, and there are no clear clusters in the underlying scatterplot. Interested readers can also refer to the web page of our project to explore this robustness experiment.

Table 3: Robustness of using machine learning model vs. using scagnostics algorithm for predicting/calculating outlying scores.

Prediction type	Scatterplot	Outlying score
Scagnostics algorithm	Year 2011	0.296
	Year 2014	0.548
Machine learning model	Year 2011	0.282
	Year 2014	0.292

5.4 Run-time evaluation

To test the performance of our approach, we used the learned predictive model to predict 589 scatterplots from the real datasets and reported the run-time, as shown in Table 4. All executions were performed on an iMac with 3 GHz 6-Core Intel Core i5, macOS Catalina Version 10.15.3, 8 GB of RAM (please refer to the web page of our project to evaluate the run-time for these tasks in reader's own device). It is worth noting that the prediction time reduces more than ten-fold (20.019 *ms* vs. 1.630 *ms*). The computational complexity of scagnostics features vary depending on the number of bins after the binning process [30]. Thus, the standard deviation of scagnostics algorithm run-time is high for different experimented datasets. However, it takes time to load a trained model (average of 86.032 *ms*), and it also takes a longer time (about 25 *ms*) to make the first prediction. The reason is that the first call includes the compilation time of WebGL shader programs for the model [27]. Thus, it is not beneficial to load and use a machine learning model to predict a single scatterplot at a time (i.e., we should use a machine model to predict many scatterplots or a batch of them).

Table 4: Run time evaluation on using machine learning model vs. using scagnostics algorithm.

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

5.5 Performance evaluation in Python

This section describes the overall performance of the two approaches (scagnostics algorithms vs. CNN models) in Python implementations. Specifically, we use the learned predictive model as described in Section 4.3 and the Python binding to R scagnostics [13] to predict/calculate the scagnostics scores for all the 7,369 scatterplots described in Section 4.1 for 31 times each. We then use *psutil* [22] to measure their performance metrics, including run-time performance, CPU percent utilization, and memory percent utilization. All executions were performed on MacBook Pro with 2.6 GHz 6-Core Intel Core i7 (12 logical cores), 16 GB 2400 MHz DDR4, macOS Catalina Version 10.15.4. Figure 9 reports these performance metrics.

Figure 9 (a) shows that the cumulative run-time (in nanoseconds) for scagnostics algorithms is significantly higher compared to that of using the CNN model. However, it is worth noting that the efficiency is due to the reason that the CNN model makes use of more memory and also more logical cores (12 in this case) for parallel computations. Specifically, Figure 9 (b) depicts that the CNN model utilizes all of the 12 logical cores. Thus its CPU percent utilization gets up to 1,200% while that for scagnostics algorithms is only about 100%. However, if we divide the CPU utilization percentages for the CNN approach by the number of logical cores (12), they will also get to approximately 100%. Finally, Figure 9 (c) unveils that the CNN model also uses more memory compared to conventional scagnostics algorithms to achieve its run-time performance.

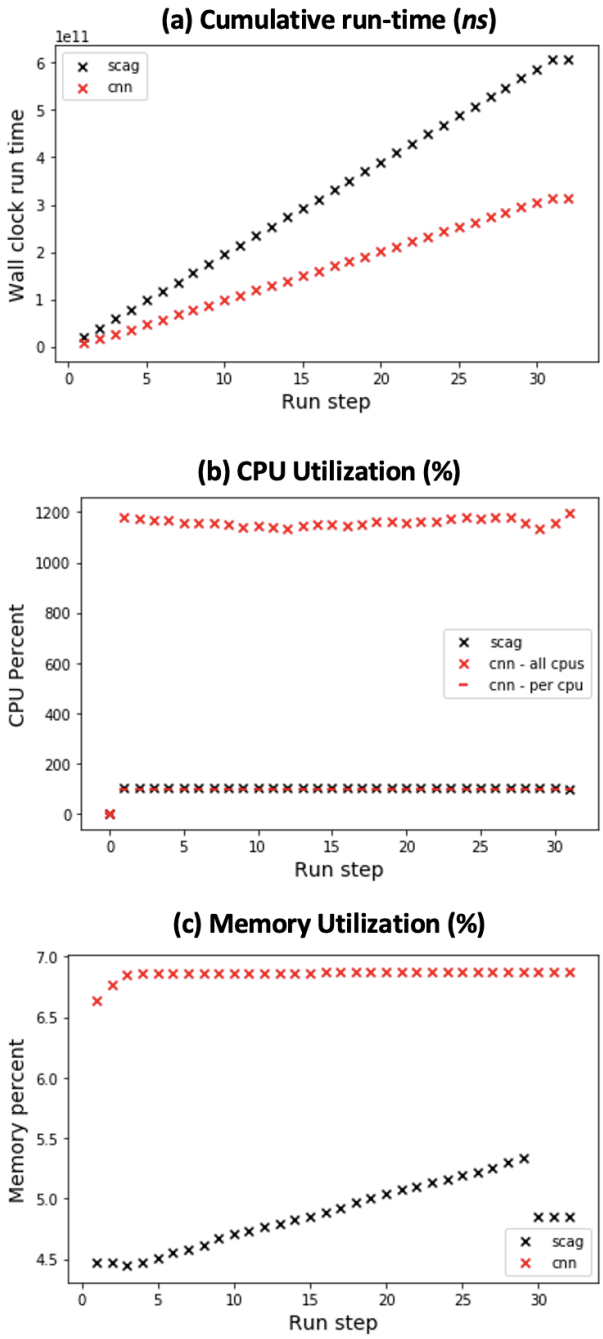


Figure 9: CNN models vs. scagnostics algorithms performance evaluations for 31 executions on 7,369 scatterplots: (a) Cumulative run-time (nanosecond), (b) CPU Percent Utilization (%), and (c) Memory Percent Utilization (%).

5.6 Implementation

The scagnostics scores are computed using *ScagnosticsJS* [20] and ScagCNN machine learning models are developed using Keras, a Python deep learning library [12]. ScagCNN is developed using JavaScript and in particular the D3.js library [7]. The python code, the online ScagCNN prototype, demo video, source code, and more examples are available on our Github repository at https://idatavisualizationlab.github.io/V/ScagCNN/.

6 CONCLUSIONS

In this paper, we implement a machine learning model, called ScagCNN, to estimate the scagnostics scores. This model aims to reduce the scagnostics computation time and sensitivity to slight changes by pre-training a CNN model on typical pair-wise distributions. We also implement an interactive web interface for exploring and verifying the predictive performance of the model and providing a visual explanation about whether a prediction is accurate and how ScagCNN come up with the estimate scores for the give scatterplots. Finally, we test the performance of our proposed approach on both synthesized and real-world datasets of various sizes. The idea has more general implications: it can be extended to the computations of other visual features such as adjacency matrix, parallel coordinates, or graph- or pixel-based visualizations. We plan these research directions in our future work.

REFERENCES

[1] Wael AbdAlmageed, Yue Wu, Stephen Rawls, Shai Harel, Tal Hassner, Iacopo Masi, Jongmoo Choi, Jatuporn Lekust, Jungyeon Kim, Prem Natarajan, et al. 2016. Face recognition using deep multi-pose representations. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 1–9.

[2] A. Asuncion and D.J. Newman. 2007. UCI Machine Learning Repository. http://www.ics.uci.edu/~mlearn/MLRepository.html

[3] M. Aupetit and M. Sedlmair. 2016. SepMe: 2002 New visual separation measures. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*. 1–8. https://doi.org/10.1109/PACIFICVIS.2016.7465244

[4] Michael Behrisch, Michael Blumenschein, Nam Wook Kim, Lin Shao, Mennatallah El-Assady, Johannes Fuchs, Daniel Seebacher, Alexandra Diehl, Ulrik Brandes, Hanspeter Pfister, et al. 2018. Quality metrics for information visualization. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 625–662.

[5] Enrico Bertini and Giuseppe Santucci. 2004. Quality metrics for 2d scatterplot graphics: automatically reducing visual clutter. In *International Symposium on Smart Graphics*. Springer, 77–89.

[6] Enrico Bertini and Giuseppe Santucci. 2006. Visual quality metrics. In *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization*. 1–5.

[7] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics* 17, 12 (2011), 2301–2309.

[8] Bureau of Labor Statistics. 2018. http://www.bls.gov/data/.

[9] Tuan Nhon Dang and Leland Wilkinson. 2014. Transforming scagnostics to reveal hidden features. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 1624–1632. https://doi.org/10.1109/TVCG.2014.2346572

[10] Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. 1983. On the shape of a set of points in the plane. *IEEE Transactions on information theory* 29, 4 (1983), 551–559.

[11] Kaggle. 2018. https://www.kaggle.com/datasets.

[12] Keras. [n.d.]. Keras: The Python Deep Learning library. https://keras.io/. Accessed: 2019–11-5.

[13] Josua Krause. [n.d.]. Python binding to R scagnostics. https://github.com/nyuvis/scagnostics. Accessed: 2020–03-24.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[15] Dy D. Le, Vung Pham, Huyen N. Nguyen, and Tommy Dang. 2019. Visualization and Explainable Machine Learning for Efficient Manufacturing and System Operations. *Smart and Sustainable Manufacturing Systems* 3, 2 (Feb. 2019), 20190029. https://doi.org/10.1520/ssms20190029

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[17] Y. Ma, A. K. H. Tung, W. Wang, X. Gao, Z. Pan, and W. Chen. 2020. ScatterNet: A Deep Subjective Similarity Model for Visual Analysis of Scatterplots. *IEEE Transactions on Visualization and Computer Graphics* 26, 3 (March 2020), 1562–1576. https://doi.org/10.1109/TVCG.2018.2875702

[18] National Consortium for the Study of Terrorism and Responses to Terrorism (START). 2018. http://www.start.umd.edu/.

[19] V. Pham and T. Dang. 2019. Outliagnostics: Visualizing Temporal Discrepancy in Outlying Signatures of Data Entries. In *2019 IEEE Visualization in Data Science (VDS)*. 29–37. https://doi.org/10.1109/VDS48975.2019.8973379

[20] V Pham and T Dang. 2020. ScagnosticsJS : Extended Scatterplot Visual Features for the Web. (2020). https://doi.org/10.2312/egs.20201022

[21] Vung Pham, Ngan Nguyen, Jie Li, Jon Hass, Yong Chen, and Tommy Dang. 2019. MTSAD: Multivariate Time Series Abnormality Detection and Visualization. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 3267–3276.

[22] psutil. [n.d.]. psutil documentation. https://psutil.readthedocs.io/en/latest/. Accessed: 2020–03-24.

[23] Andrea Saltelli, Stefano Tarantola, Francesca Campolongo, and Marco Ratto. 2004. *Sensitivity analysis in practice: a guide to assessing scientific models*. Vol. 1. Wiley Online Library.

[24] Michael Sedlmair and Michaël Aupetit. 2015. Data-driven evaluation of visual quality measures. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 201–210.

[25] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[27] TensorFlow. [n.d.]. Tensorflow documentation. https://github.com/tensorflow/tfjs/tree/master/tfjs-converter. Accessed: 2019–12-24.

[28] Visual Geometry Group. [n.d.]. Visual Geometry Group. http://www.robots.ox.ac.uk/~vgg/. Accessed: 2019–11-4.

[29] Y. Wang, Z. Wang, T. Liu, M. Correll, Z. Cheng, O. Deussen, and M. Sedlmair. 2020. Improving the Robustness of Scagnostics. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (Jan 2020), 759–769. https://doi.org/10.1109/TVCG.2019.2934796

[30] Leland Wilkinson, Anushka Anand, and Robert Grossman. 2005. Graph-theoretic scagnostics. *Proceedings - IEEE Symposium on Information Visualization, INFO VIS* (2005), 157–164. https://doi.org/10.1109/INFVIS.2005.1532142

[31] L. Wilkinson, A. Anand, and R. Grossman. 2006. High-Dimensional Visual Analytics: Interactive Exploration Guided by Pairwise Views of Point Distributions. *IEEE Transactions on Visualization and Computer Graphics* 12, 6 (2006), 1363–1372.

[32] Leland Wilkinson and Graham Wills. 2008. Scagnostics distributions. *Journal of Computational and Graphical Statistics* 17, 2 (2008), 473–491.

[33] World Bank Open Data. 2018. https://data.worldbank.org/.

[34] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. 2019. *Dive into Deep Learning*. https://d2l.ai. Accessed: 2019-11-4.