Deep Colormap Extraction from Visualizations

Lin-Ping Yuan, Wei Zeng, *Member, IEEE*, Siwei Fu, *Member, IEEE*, Zhiliang Zeng, Haotian Li, Chi-Wing Fu, *Member, IEEE*, and Huamin Qu, *Member, IEEE*

Abstract—This work presents a new approach based on deep learning to automatically extract colormaps from visualizations. After summarizing colors in an input visualization image as a Lab color histogram, we pass the histogram to a pre-trained deep neural network, which learns to predict the colormap that produces the visualization. To train the network, we create a new dataset of ∼64K visualizations that cover a wide variety of data distributions, chart types, and colormaps. The network adopts an atrous spatial pyramid pooling module to capture color features at multiple scales in the input color histograms. We then classify the predicted colormap as discrete or continuous, and refine the predicted colormap based on its color histogram. Quantitative comparisons to existing methods show the superior performance of our approach on both synthetic and real-world visualizations. We further demonstrate the utility of our method with two use cases, *i.e.*, color transfer and color remapping.

Index Terms—Color extraction, information visualization, deep learning, color histogram

1 Introduction

Color is a fundamental visual means to convey information and knowledge in data. Colormaps are used to characterize the mapping from data domain to colors in visualizations [7], [46]. For instance, colors can help depict the attributes in geographical data such as income and dominant commercial sectors [19]. However, designing effective colormaps has always been challenging, since many factors, such as data types, tasks, goals, and users [46], [41], [32], need to be taken into account simultaneously.

Many color usage guidelines and optimization methods have been proposed to facilitate color design; see [41], [32], [61] for systematic reviews. Early studies suggested various standards, e.g., a color design should consider the color properties such as uniformity [37] and sequence [50], as well as human factors such as perceptual discrimination [21]. These guidelines later became fundamentals for *rule-based* colormap selection [61]. Besides these factors, data and tasks should also be carefully considered in the color design process [61], such as for tree-structure data [45] and categorical data [16], to improve the semantic reasoning [40], [27].

The need for colormap design is immense. Many colormaps proposed by the above works have been integrated into existing visualization tools [61]. Nevertheless, while inappropriate color ranges could hinder the exploration and analysis of features in data [61], problematic color assignments, which disregard human perception, e.g., rainbow colormaps [6], are still ubiquitous. One promising approach to overcome the issues is to re-style the visualizations [25], [35]. To achieve this, however, requires identification of the color usages in the given visualizations. Poco et al. [36] developed a semi-automatic approach to extract colormaps from visualization images. The results benefit many usage scenarios, e.g., to optimize poorly-designed colormaps and to enable interactive overlays to improve readability. However, the method relies on detecting organized color legends in the visualizations, which may not be available, particularly for those found from the Internet [56]. Thus, some works suggested to directly extract colors from visualizations, e.g., by finding a long path of smoothly-varying colors in Lab color space for recovering continuous colormaps [56], or by adapting k-means clustering to find discrete colormaps [8], [59].

However, these methods are heuristic-based and rely on cumbersome predefined parameters, which are prone to fail for general visualizations. In this work, we present a learning-based approach to automatically extract colormaps from visualizations via a deep neural network. Instead of adapting the parameters manually, we feed a neural network with a vast amount of visualization images, and let the network learn to model an optimal mapping from the given visualizations to the original colormaps.

An immediate challenge to the approach is the lack of suitable training data. We hereby prepare a dataset by synthesizing diverse visualization images, with careful consideration of data distributions, chart types, and colormaps (Sec. 4.1). Also, we find that directly mapping from visualization images to colormaps is variant to geometric transformations such as rotation. To overcome the limitation, we first summarize an input visualization image as a color histogram in Lab color space (Sec. 4.2.1), then adopt an image-translation convolutional neural network (CNN) to produce a fixed-size color legend as an intermediate result (Sec. 4.2.2). Specifically, the network employs an atrous spatial pyramid pooling (ASPP) module [9] that adjusts the filter's field-of-view, being able to capture color features at multiple scales in the input color histogram. We further refine the result using DBSCAN or Laplacian eigenmaps to generate the final discrete or continuous colormap (Sec. 4.2.3), respectively. We conduct quantitative

- *L.-P. Yuan, H. Li, and H. Qu are with the Hong Kong University of Science and Technology. e-mail: {lyuanaa, haotian.li}@connect.ust.hk and huamin@cse.ust.hk.*
- *W. Zeng (corresponding author) is with Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. e-mail: wei.zeng@siat.ac.cn.*
- *S. Fu is with Zhejiang Lab. e-mail: fusiwei339@gmail.com.*
- *Z. Zeng, and C.-W. Fu are with the Chinese University of Hong Kong. e-mail: {zlzeng, cwfu}@cse.cuhk.edu.hk.*

Manuscript received April 19, 2005; revised August 26, 2015.

comparisons to existing methods on an evaluation dataset comprised of synthetic and real-world visualizations. The experimental results demonstrate superior performances of our approach and prove its effectiveness and robustness (Sec. 5). We further show two applications of our approach: (i) color transfer and (ii) color remapping (Sec. 6). The synthetic visualization image corpus and deep neural network code are available at https://bit.ly/3bjMdyV.

2 Related Work

Color design for visualization. Designing effective colormaps for visualizations requires both experience and a good sense of aesthetics. Many research efforts have been devoted to the problem, of which systematic reviews are presented in [41], [32], [61]. Among these studies, ColorBrewer [19] provides widely-used colormaps. While the proposed colormaps have greatly improved the color design in maps and many other applications, the color mapping might miss interesting features in data due to inappropriate color ranges [61]. This promotes the considerations of data and tasks when designing colormaps. For instance, Lee et al. [26] showed that the conventional ColorBrewer palette could produce limited visibility of categorical differences in maps, so they improved the color assignment by optimizing the class visibility when presenting coherent categorical structures. Other factors such as contrast [31] and visual separability [48] have also been studied to maximize data visualization efficiency. Furthermore, associating colors with resonant semantics helps discriminate among the data values [27], [40].

Color design and optimization are out of the scope of this work. Instead, we focus on exemplar-based visualization design, which has been successfully demonstrated in applications such as image colorization [24], [57], multiple-view visualization design [11], and graph layout tuning [34], [44]. We explore reverse-engineering approaches to extract colormaps from given visualizations, and also develop two applications that are built upon our method: (i) transferring color design from source to target visualizations [52], [24], and (ii) remapping color coding coherently with the underlying data distribution [46].

Color extraction from images. Extracting colors from images is a basic operation in many applications, e.g., image recoloring, decolorization, etc. Irony et al. [24] identified the colors in a segmented image sample, and applied the colors to colorize grayscale images. A typical approach here is to cluster the image colors in the RGB color space then identify the most prominent ones [8], [59]. Another approach forms a convex hull over the image colors in the color space, and applies geometric processing techniques to extract the primary colors [43], [42]. Both approaches share a common step of summarizing the colors in a certain color space to improve the robustness for images of arbitrary size.

These works, however, target mostly natural images rather than visualizations. Hence, color design principles for visualizations are not considered, e.g., color ordering for representing ordinal data. For instance, Chang et al. [8] employed a simple policy of sorting all the colors according to the luminance. In contrast, many existing colormaps order the colors based on their hues. Yoo et al. [56] addressed the ordering problem by fitting a long curve in the CIELab color space to smoothly pass through the colors in a given visualization. Unfortunately, the method only works for continuous colormaps, *not* applicable to discrete ones. Even for continuous colormaps, the method's performance drops when the underlying data distribution is uneven (see evaluations in Sec. 5). Recently, Poco et al. [36] developed a semi-automatic approach to identify color legends in visualizations, classify the legends as discrete or continuous colormaps, and then apply image segmentation techniques to extract the colors from the legends. Although the method can handle both continuous and discrete colormaps, it assumes the input visualization contains an explicit color legend. Yoo et al. [56] found that only 29% of 611 web visualization images have a proper color legend. Image queries on Google search engine with keywords 'data visualization', and 'information visualization' also show similar results; see Supplementary Fig. 1 for examples.

In summary, prior heuristic-based approaches suffer from tedious parameter settings, such as different numbers of discrete colors [8], and color order recovering [56]. To overcome the limitations, we develop a learning-based approach, where we prepare a dataset with careful considerations on the data distributions, chart types, and colormaps to supervise the network training.

Deep learning for data visualization. Advancements of deep learning in various research areas, particularly in computer vision and image understanding, have inspired the visualization community to address various problems; see [47], [53] for recent surveys. For example, Jung et al. [25] trained a deep neural network to classify chart types. Chen et al. [12] improved the effectiveness and robustness in selection of 3D point clouds using deep learning. Zhao et al. [60] recommended charts for exploratory visual analysis using an interactive system coupled with machine intelligence. Some researches further modeled perception-related problems using deep learning. Wu et al. [54] adopted a Siamese neural network to assess chart layout qualities from pairwise comparison data, while Haehn et al. [17] evaluated various neural networks in analyzing graphical elements. Besides, some other recent works aim for understanding and diagnosing the learning process in deep neural networks through visual analytics [28], [22].

Adding on to this line of deep learning for visualization, we develop and train the first deep neural network for colormap extraction from visualization images. Compared with existing heuristic-based colormap extraction methods, our deep model shows better generalizability and can robustly handle more variety of information visualization images and colormaps.

3 Overview

This section clarifies the scope of the work (Sec. 3.1), summarizes the problem statement (Sec. 3.2), and overviews our method (Sec. 3.3).

3.1 Method Scope

Recovering visual encodings from visualizations has recently gained much attention in the visualization commu-

nity [35]. Some studies have exploited structures in interactive D3 visualizations in SVGs [18], [23]. Instead, we constrain the scope of this work to static bitmap images, which are the most commonly available format for charts, and accurate extraction remains challenging [18]. More specifically, we focus on information visualization, or abstract two-dimensional data visualization, e.g., bar charts, maps, and scatter plots, etc. Visualizations for scientific data (e.g., particles, streamlines, and volumes) usually depict depth or motion with transparency, which, however, may distract the color saturation and luminance [32]. We leave it as a future work to extract colormaps from scientific visualizations.

In the early stage of this work, we conducted literature surveys on color usage in information visualization. Based on assorted systematic reviews [41], [32], [61], we further make careful specifications on the scope of this work:

- *1D v.s. 2D colormaps*. A variety of 2D colormaps have been proposed for visualizing bivariate or multivariate data. However, they were criticized due to the difficulty in data interpretation, especially when both attributes have multiple levels [4], [32]. In this work, we only consider 1D colormaps, which are the most commonly-used colormaps in the production of information visualization [61].
- *Blending* or *interpolation*. When visualizing multivariate data, some frequently-used techniques are color blending or interpolation [13]. Both techniques will generate nonlinear mappings of colors, which may hinder the deep neural network to learn the underlying colormaps.
- *Background & text colors*. The presence of dissimilar colors near one another can greatly affect the color perception [30]. To create effective colormap extraction, background and text colors should be removed before processing.
- *Linear v.s. non-linear color scales*. There are some perceptually non-linear color scales that allot more colors to data of interest and fewer colors to all other data, to cope with skewed data distributions [46]. Such non-linear interpolation between data and colors can cause ambiguity with commonly available linear color scales. Hence, this work regards all color scales behind input visualizations as linear. That is, we do not extract how data values map to colors.

3.2 Problem Statement

We classify colormaps as *discrete* or *continuous*. A discrete colormap contains only a small set of colors, while a continuous colormap is essentially a curve in a color space. We model both discrete and continuous colormaps as an ordered list of m color values, denoted as $C := \{c^i\}_{i=1}^m$, where i indicates the i-th color in C and m is small, typically less than 10 for discrete colormaps [51], and we fix it as 256 for continuous colormaps. The setting is pragmatic, as continuous colormaps are usually presented as a 1D image with smoothly-varying colors.

In information visualizations, colormaps are employed to map data values to colors. We treat the original colormap as the ground truth $C_{gt} := \{c_{gt}^i\}_{i=1}^m$ and aim to develop a new algorithm to automatically extract colormap $C_{out} := \{c_{out}^j\}_{j=1}^n$ that is similar to C_{gt}. Notice that C_{out} may not have the same number of colors as C_{gt}. We consider the following requirements for such an algorithm:

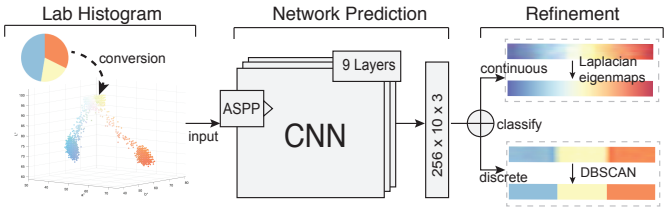


Fig. 1. Our approach consists of three stages: (i) histogram conversion, (ii) network prediction, and (iii) refinement.

- *Effectiveness.* The algorithm should generate C_{out} in color space S, and C_{out} should be similar to C_{gt}, meaning that

$$\min_{C_{out} \in S} D(C_{gt}, C_{out}) \,, \tag{1}$$

  where D is a distance function. Specifically, we should preserve the *ordering* of colors in each colormap, which is essential for ordinal data. To this end, we employ dynamic time warping (DTW) that is a popular technique for comparing sequences [14], to model D in Eq. 1; see Sec. 5.2 for details.
- *Robustness.* The method should (i) be applicable to both discrete and continuous colormaps; (ii) handle visualizations with or without given color legends; (iii) deal with various types of information visualizations, such as bar charts, pie charts, and line plots; and (iv) be invariant to geometric transformation such as rotation.

3.3 Technique at Large

Extracting colormaps from given visualizations is a reverse-engineering process. In contrast to prior heuristic-based methods, we approach the problem from a new perspective, by regarding the process as a mapping problem, *i.e.*, to map a visualization to the associated colormap. We hereby adopt and train a deep neural network model, and drive it to learn with carefully-prepared example visualizations paired with the underlying colormaps.

As shown in Fig. 1, our approach has three major stages:

1) *Histogram conversion* (Sec. 4.2.1). Preliminary experiments reveal that directly mapping from visualization images to colormaps is highly variant against rotation (Fig. 3). To facilitate the network to learn the color features instead of other visual primitives such as shapes and locations, we first summarize the input visualization image into a color histogram in the Lab color space. This conversion can effectively reduce the number of variants in the network input, and ensure the network is rotation invariant.
2) *Network prediction* (Sec. 4.2.2). The network takes a color histogram as input and generates a fixed-size image as its output. Nevertheless, there exist several color-related issues, e.g., null features in the input histograms. We make adaptations to the baseline

ResNet18 network [20], including the incorporation of the ASPP module [9], to tackle the challenges.

3) *Refinement* (Sec. 4.2.3). Lastly, we refine the output network prediction and produce an ordered list of color values. The refinement is a two-fold process. First, we utilize a binary classification rule to determine if the prediction is a continuous or discrete colormap. Second, we apply Laplacian eigenmaps or DBSCAN to extract the output continuous or discrete colormaps, respectively.

4 Deep Colormap Extraction

To train a deep neural network, we first prepare training data that covers a wide variety of information visualizations, each paired with a corresponding colormap (Sec. 4.1). Next, we present the technical details in our deep colormap extraction method (Sec. 4.2).

4.1 Data Generation

While some visualization corpora are available (e.g., [38], [5], [25], [36]), they do not provide the paired colormaps. It would require tremendous labor work to manually reconstruct the underlying colormap for each given visualization. Motivated by the use of image synthesis to generate training datasets [29], [17], we develop an automatic method to synthesize information visualizations with corresponding colormaps. Thanks to the wide usage of visualization toolkits like D3 in real-world applications, the synthetic data can mimic real-world visualizations [35].

We next elaborate on the considerations for maximizing the coverage of the sample information visualizations. Then, we further present the technical details of synthesizing a new dataset, which contains ∼64K visualizations with appropriate combinations of realistic data distributions, chart types, and colormaps.

Design considerations. Visualization creation can be viewed as an exploratory process in the *control* space (which includes the visualization styles, layout, colormaps, etc.) until satisfactory *visualization* results are produced for the input *data* [10]. That is to say, a visualization depends both on the input data and the control parameters. Specifically, we diversify the control parameters in two aspects: (i) chart types produce variants in the geometric space, e.g., lines *vs.* areas, and (ii) colormaps complement the geometric diversity in the color space.

Fig. 2 illustrates how we produce diverse visualizations, for example, through 2 data distributions (normal and beta) × 2 chart types (pie and choropleth map) × 6 colormaps (3 discrete and 3 continuous) = 24 visualizations. Among the combinations, some are non-practical, e.g., pie charts with continuous colormaps. Nevertheless, even without such combinations, there are over 10 combinations remaining, four of which are presented in Fig. 2 (bottom).

Implementation details. To enrich the dataset coverage, we first create vast variants in the following aspects.

- *Data distributions*. To cope with real-world scenarios, we first select 1,041 datasets with attribute number in the range of [1, 10] from the Rdatasets[1], which contains over 1,300 popular datasets used by statisticians and visualization researchers, e.g., the famous *Iris* flower data; see Supplementary Fig. 2 for more details. Second, we enrich the dataset with synthetic one- and two-dimensional data. One-dimensional data follows the common distributions such as normal, beta, and Poisson distributions, while two-dimensional data can either be independent or follow a linear or inverse-linear correlation. For each distribution type, we create 10 sets of random parameters, yielding a set of data distribution variants. All these data distributions are generated using SciPy[2].
- *Chart types*. We first study common charts as identified by prior researches, e.g., Revision [38] and ChartSense [25]. Among these charts, Venn diagrams and radar charts usually incorporate translucency in colors, which are ignored for now. We also include other chart types such as heatmaps and choropleth maps, resulting in a total of eight chart types, *i.e.*, {*line chart*, *pie chart*, *grouped bar chart*, *stacked bar chart*, *scatter plot*, *stream graph*, *heat map*, and *choropleth map*}. For each chart type, we further consider different parameter settings, such as line width in line charts, and spacing between bars in bar charts, by referencing to common visualization libraries, including D3 and Vega, and applications such as Tableau and MS Excel.
- *Colormaps*. We look for frequently-used colormaps from various visualization libraries, including ColorBrewer [1], colorcet [2], and D3 [3]. We constrain the number of colors to be three to ten for discrete colormaps, since such number is already sufficient to cover most visualizations of discrete data; see Supplementary Fig. 2. Duplicates from different sources are manually removed after approximate comparisons. In the end, we collect 236 distinct colormaps, among which 54 are continuous, and the remainings are discrete. For each colormap, we create an ordered

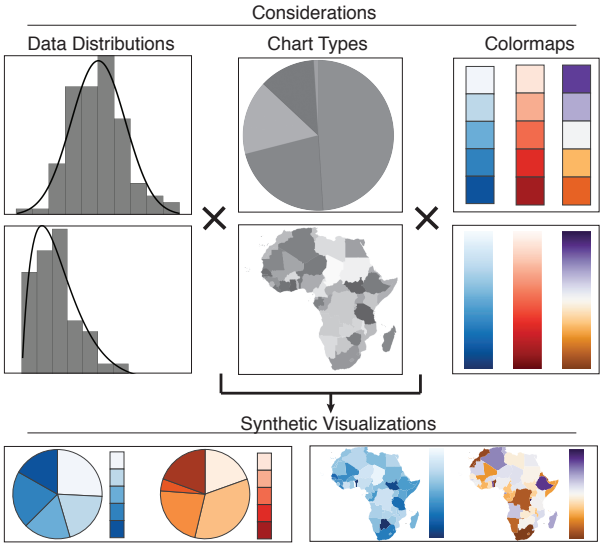


Fig. 2. We carefully consider various variants in data distributions, chart types, and colormaps (top), when generating the synthetic visualizations with the corresponding colormaps (bottom).

1. https://vincentarelbundock.github.io/Rdatasets/

2. https://www.scipy.org

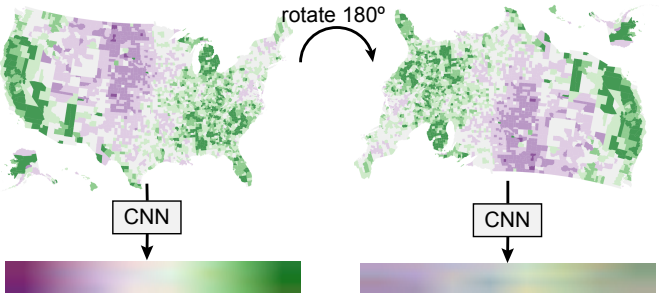


Fig. 3. A neural network that directly learns colormaps from visualization images is variant to image rotation.

list of Lab colors for final quantitative evaluation, and a 10×256 colormap image derived from the color list to supervise the network training.

For each rational combination of these variants, we render one visualization using D3, and save it using a backend server. In this way, we generate a total of 63,965 visualization images. All images are in the size of 512 × 256.

4.2 Building Deep Colormap Extraction Model

With the synthetic dataset, we can proceed to the next phase of learning the mapping from visualizations to colormaps. We formulate a new deep colormap extraction model that consists of three main stages: *histogram conversion* (Sec. 4.2.1), *CNN prediction* (Sec. 4.2.2), and *refinement* (Sec. 4.2.3).

4.2.1 Histogram Conversion

A vital requirement of neural networks for visualization image analysis is to preserve the geometric invariance, like rotation and scale [17]. We conducted preliminary experiments of training a CNN model to directly map visualization images to colormaps. However, results showed that the model cannot fulfill the requirement of being rotation invariant, as demonstrated in Fig. 3. To eliminate the effects by geometric features, *e.g.*, shapes and positions, we opt to summarize colors in an input visualization image as a 3D histogram in the CIELab color space, then convert the 3D histogram into a 2D map format that is preferable for existing CNN architectures. Here, the CIELab color space is chosen because of its perceptual uniformity. The conversion works as follows:

1) *Construct color histograms*. We divide the L channel of the CIELab color space into 256 bins, and a and b channels into 128 bins, yielding two 256×128 color histograms: one for *L-a* and the other for *L-b*. Here, each histogram entry stores the number of pixels in the input visualization image that has the corresponding *L-a* or *L-b* values.
2) *Filter-and-normalize*. Next, we filter out the colors for background, text, and grid elements by zero-ing the histogram entries that correspond to these colors. For the background color, we traverse pixels at the visualization image borders and look for a dominant color that occupies $\geq 80\%$ of all these pixels. For text and grid element colors, we make a similar assumption as [36] that they are black. Notice that these operations filter out colors that are rare in common colormaps, even though some colors may seem the same (*e.g.*, dark gray *vs.* black). Then we normalize all entries in the *L-a* and *L-b* histograms in the range [0,1] by dividing the maximum value.
3) *Concatenation*. Lastly, we concatenate the two histograms into a single 2D map of size 256×256 as the network input. Essentially, this is a single-layer 2D map that summarizes the color information in the input visualization for the network to learn to map it to the target colormap.

4.2.2 CNN Prediction

Network architecture. We build the network based on ResNet18 [20]. The overall architecture is shown in Fig. 4. First, we adopt four stages of convolutional layers. Each stage has two convolutional layers, each activated by a ReLu function and using a fixed 3×3 kernel size. After each stage, the feature map size is halved and the depth doubles. Through this hierarchical structure, the network is able to progressively distill the features. Furthermore, we adopt the ASPP module [9], after each stage of convolutional layers. Since, most values (the given color histogram) in the network input are zeros, the null values hinder the network to learn effectively, and may even cause it to collapse, i.e., poorly converge. Hence, we replace the fixed-size kernel with a pyramid kernel provided the ASPP module; by using a larger kernel, the network can have a larger receptive field to distill larger-scale (more nonlocal) features. After such feature learning, we employ a convolutional layer with a 1×1-kernel to further process the extracted features. Lastly, we generate the output colormap from the last layer through a sigmoid function, and bilinearly resize the output to produce a colormap of size 10×256×3. By modeling the output as a fixed-size image, our network can effectively handle the conflict of limited neurons *vs.* enormous colors.

Loss function. To supervise the network learning, we use an L_2 loss to measure the distance between the network output and the ground truth colormap, since both are simply 2D images by nature:

$$Loss_{L2}(X,Y) = \frac{1}{n}\sum_{i=1}^{n} ||x_i - y_i||^2 \,, \tag{2}$$

where x_i and y_i are the colors of the i-th pixel on the network output X and the ground truth colormap Y, respectively. The pixel distance is measured as Euclidean distance in Lab color space, where all channels are normalized in the range [0, 1].

Implementation details. We synthesize 63,965 visualizations in total, where we use 90% for training and 10% for testing. The network is implemented using PyTorch and trained on a workstation with NVIDIA GeForce GTX 1080 Ti GPU. We employ the Adam optimizer to train the network for 80,000 iterations with a learning rate of 0.0001, and a batch size of 8. We do not use any batch normalization nor layer normalization.

Ablation analysis. We perform an ablation analysis on the model loss over training iterations to evaluate the effectiveness of modeling the Lab histogram as a 2D map

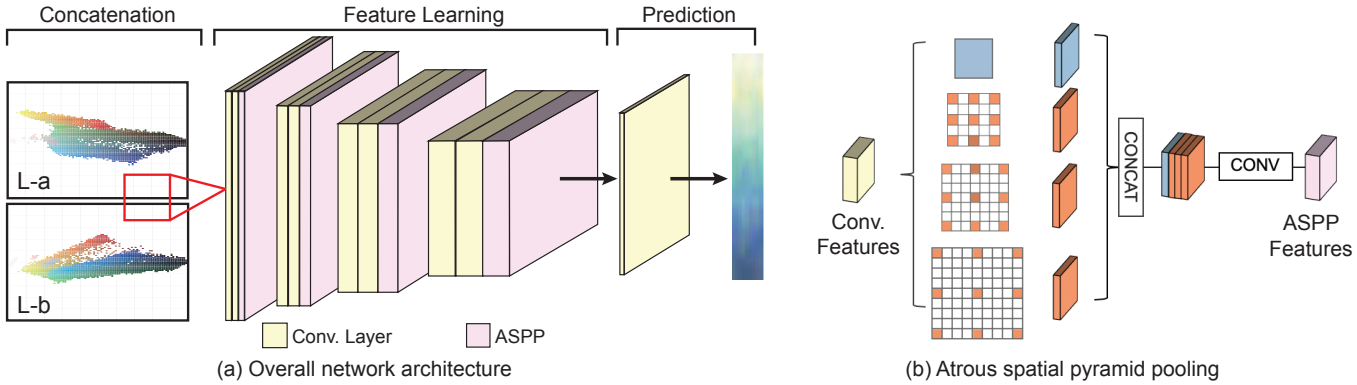


Fig. 4. The architecture of the deep neural network we adopted is presented in (a). We adopt four stages of convolutional layers (yellow boxes) for feature extraction and feature learning, and make use of ASPP modules (pink boxes) after each stage to further distill the features using a pyramid kernel. After that, to predict the output colormap, we use a 1×1 kernel-size convolutional layer and a bilinear resize to generate an image output, which is then resized to generate the output colormap. The details of the ASPP module is shown in (b).

Fig. 5. Ablation analysis of comparing network input as 3D tensor (green) *vs.* 2D map concatenation without ASPP module (orange) *vs.* 2D map concatenation with ASPP module (blue).

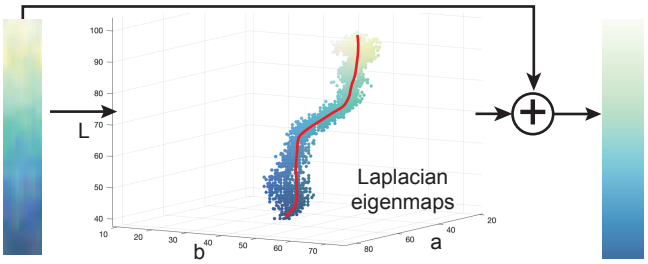


Fig. 6. Refinement workflow: CNN prediction (left), Laplacian eigenmaps for continuous colormaps (middle), and final result (right).

concatenation, and the ASPP module. Fig. 5 presents the comparison results: green curve for network input as 3D tensor without ASPP, orange curve for network input as 2D map concatenation without ASPP, and blue curve for a network with ASPP. All other hyperparameter settings employed in the networks are the same, including the training data, loss function, and learning rate, etc. We measure L_2 loss using 100 testing cases at every 100 training iterations for the first 10,000 iterations.

First, instead of using a 2D map concatenation, we can feed into the network a 3D tensor that directly represents the Lab histogram. We notice that the performance here is not comparable with 2D map concatenation. We suspect this is because general CNNs consume 3D tensor as stacks of 2D feature maps, rather than geometric features in the 3D space of the Lab histograms. Second, a comparison between the orange and blue curves shows that the network with the ASPP module can converge faster and achieve better result: the loss is twice in the network without ASPP (2.4×10^{-3}) than with ASPP (1.1×10^{-3}) after iteration 10,000.

Our current implementation of the network can be regarded as a regression function. Alternatively, we can model the problem as a multi-class classification task that directly categorizes input histograms into classes of colormaps. We train a classification network with the same input and internal structure of that in Fig. 4, but change the loss function to a multi-class cross entropy loss with 236 classes of colormaps. However, for real-world visualization, we find the classification network can easily misclassify discrete and continuous colormaps or predict colormaps with wrong color schemes, and can never recognize unseen colormaps. Our regression approach is more robust and generalizable.

More results on ablation analyses are discussed in Supplementary Sec. 3.

4.2.3 *Refinement*

Fig. 6(left) presents a network prediction of YlGnBu continuous colormap, which however is rather noisy. We refine the prediction to the final result (Fig. 6(right)):

1) *Binary classification.* We check if the prediction colormap is discrete or continuous. This can be done using a simple binary classification rule: we first aggregate histograms of the prediction result in Lab color space and normalize the histograms into the range [0, 1]. Then we filter out all bins with a normalized value smaller than 0.01. After that, we empirically classify the prediction colormap as discrete or continuous based on the number of remaining histograms.
2) *Color extraction.* Next, we apply DBSCAN and Laplacian eigenmaps to the filtered discrete and continuous colors, respectively. For DBSCAN, we experimentally find that $minNum = 4$ and $\varepsilon = 0.05$ yield good results for clustering discrete colors. In each cluster, the color with maximum histogram is chosen as the prominent color. For Laplacian

eigenmaps, we closely follow the procedures of dimension reduction as in [56].

3) *Ordering recovery*. Both DBSCAN and Laplacian eigenmaps algorithms extract unordered colors. In the last step, we recover color order by referring to the prediction results that store the ordering information. To do so, we find all pixels that have the same color as those extracted from the previous step, then measure the average pixel position with respect to prediction results for each extracted color. Finally, we sort the extracted colors in ascending order of the average pixel position.

5 Evaluation

We evaluate the *effectiveness* and *robustness* of our deep color extraction model. The evaluation is conducted on a testing dataset with both synthetic and real-world visualizations (Sec. 5.1). We present advancements of our approach over existing methods using quantitative metrics (Sec. 5.2) and representative examples (Sec. 5.3). Last, we analyze probable reasons for the advancements (Sec. 5.4).

5.1 Preparation of Testing Dataset

We extract 10% of the synthetic visualizations for evaluation, yielding 6,396 pairs of visualizations and colormaps. The visualizations cover all the colormaps and chart types we used for training; see Sec. 4.1. We denote these visualizations as **synthetic testing** in the following.

We also collect additional visualization from the Internet to further evaluate our approach on real-world visualizations. The collection process consists of two steps:

- *Visualization Collection*. We first crawl over 1000 visualizations from Google Images searching engine that covers media, websites, and government reports. Then, we filter out visualizations out of the work scope described in Sec. 3.1, and obtain 229 valid visualizations. The resulting visualizations cover both the chart types in the synthetic visualizations and others like radial plots, sunbursts, contours, etc.
- *Color Labeling*. To support quantitative evaluation, we manually label all the collected charts to serve as ground truth. All visualizations using continuous colormaps have legends, over 90% of which are explicitly presented in the charts and the rest can be contained from the source websites. For visualizations using discrete colormaps, we retrieve individual colors from explicit legends or the charts themselves, and store the colors in order.

We collect in total 229 real-world visualizations, among which 126 visualizations use colormaps in the training dataset (denoted as **real seen**), whilst colormaps of the remaining 103 visualizations are not in the training dataset (denoted as **real unseen**). We conduct experiments on the testing dataset, and report the results by categories of **synthetic testing, real seen, and real unseen**, respectively.

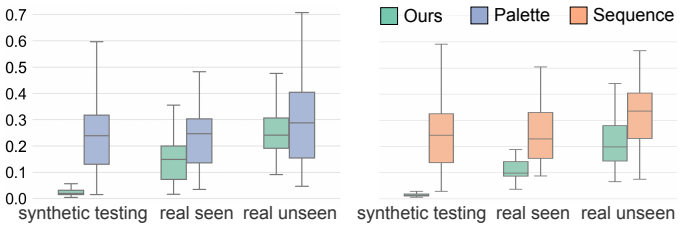


Fig. 7. Comparing D_{dtw} on synthetic testing, real seen, and real unseen visualizations. As indicated by the smaller D_{dtw} values, our method achieves better performances on all three categories of visualizations than the baseline techniques (Palette [8] for discrete colormaps (left), and Sequence [56] for continuous colormaps (right)), especially for the synthetic testing visualizations.

5.2 Quantitative Comparison

Quantitative metric: We model both continuous and discrete colormaps as an ordered list of colors $C := \{c^i\}_{i=1}^m$ (Sec. 3.2). To account for color ordering, we employ dynamic time warping (DTW) to calculate the distance (denoted as D_{dtw}) between the ground truth colormap $C_{gt} := \{c_{gt}^i\}_{i=1}^m$ and the output colormap $C_{out} := \{c_{out}^j\}_{j=1}^n$. DTW is a commonly-used similarity measuring algorithm for two series, since the algorithm is insensitive to local compression and stretches, and can optimally deform one of the two input series onto the other [14].

To compute D_{dtw}, we first generate a coupling between C_{gt} and C_{out}, which is a set L of pairings $\{\delta_k\}_{k=1}^K$, where $\delta_k = (c_{gt}^k, c_{out}^k) \in [m] \times [n]$. We can compute:

$$D_{dtw}(C_{gt}, C_{out}) = \min_{L} \sum_{\delta_k \in L} d(c_{gt}^k, c_{out}^k), \tag{3}$$

where $d(c_{gt}^k, c_{out}^k)$ is the Euclidean distance of c_{gt}^k and c_{out}^k in the normalized Lab color space (all channels are normalized to [0, 1]). Smaller D_{dtw} indicates better performance. We adopt the *dtw* package [14] to compute D_{dtw}.

Baseline techniques: We compare the performance of our approach with two existing methods for colormap extraction, *i.e.*, palette-based [8] and sequence-preserving [56]. We omit the legend-based approach [36] on synthetic visualizations, as the method requires explicit color legends in the image space that are not available in our synthetic visualizations. On the other hand, both palette-based [8] and sequence-preserving [56] methods, as well as ours, process visualization images in the color space. Nevertheless, a quantitative comparison with [36] on real-world visualizations with explicit legends can be found in the Supplementary Sec. 4.2. For a fair comparison, we only evaluate the palette-based method [8] on discrete colormaps, and the sequence-preserving method [56] on continuous colormaps. Specifically, we re-implemented sequence-preserving method [56] in MATLAB, and employed an online re-implementation [3] of the palette-based method [8].

Results. Fig. 7 presents the normalized D_{dtw} measured on three categories of synthetic testing, real seen, and real unseen visualizations. We first analyze the performance of our method across the three categories, and note that D_{dtw} varies. Our method achieves the best performance on

3. https://github.com/b-z/photo_recoloring

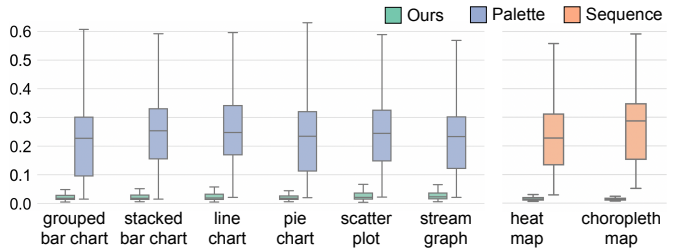


Fig. 8. Comparing D_{dtw} on the synthetic testing visualizations of different chart types. Our method achieves better performances in all chart types, as indicated by the smaller D_{dtw} values.

synthetic testing visualizations, for both discrete (left) and continuous (right) colormaps. For real seen visualizations, the D_{dtw} values increase, indicating the quality of predicted colormaps reduces. There can be various reasons behind this, such as opaque or semi-transparent texts, low resolutions, and noise introduced to their color histograms. The performance is better than that of real unseen visualizations, which have the highest D_{dtw} values. Since our approach is data-driven and the colormaps are unseen, it is not surprising that the performance drops.

Next, we compare our method with two baseline techniques across the three visualization categories. We conduct the independent *t*-tests to examine the statistical significance among different methods. Specifically,

- *Synthetic testing.* Our method significantly outperforms both palette-based ($t = -91.3, p < 0.0001$) with mean D_{dtw} reduced from $0.250_{[0.245,0.254]}$ (denoting mean with 95% CIs) to $0.029_{[0.028,0.030]}$ (88% improvement) for discrete colormaps, and sequence-preserving ($t = -50.1, p < 0.0001$) method with mean D_{dtw} reduced from $0.244_{[0.235,0.253]}$ to $0.015_{[0.014,0.016]}$ (94% improvement) for continuous colormaps.
- *Real seen.* Our method still achieves significant smaller D_{dtw} than both palette-based ($t = -5.1, p < 0.0001$) and sequence-preserving ($t = -8.1, p < 0.0001$) method. Yet, the improvements drop to 38% (mean D_{dtw} reduced from $0.240_{[0.212,0.269]}$ to $0.149_{[0.128,0.171]}$) for discrete colormaps, and to 54% (mean D_{dtw} reduced from $0.227_{[0.192,0.263]}$ to $0.104_{[0.093,0.115]}$) for continuous colormaps.
- *Real unseen.* No significant difference is observed between ours and palette-based method ($t = -1.6, p = 0.12$), whilst ours still outperforms sequence-preserving method with mean D_{dtw} reduced from $0.332_{[0.283,0.380]}$ to $0.206_{[0.172,0.241]}$ (38% improvement) for continuous colormaps.

Finally, we further compare three methods on synthetic testing visualizations in a finer-grained level by breaking down the results according to chart types, as shown in Fig. 8. Overall, our approach outperforms both palette-based and sequence-preserving methods. In more detail:

- *Discrete colormaps.* Our method achieves significantly smaller D_{dtw} than the palette-based method for all six chart types with discrete colormaps ($t = -91.3, p < 0.0001$). More specifically, mean D_{dtw} is reduced from $0.218_{[0.208,0.229]}$ to $0.026_{[0.024,0.028]}$ for *grouped bar chart* (88.1% improvement), from $0.263_{[0.251,0.275]}$ to $0.029_{[0.027,0.031]}$ for *stacked bar chart* (89.0% improvement), from $0.260_{[0.248,0.273]}$ to $0.029_{[0.027,0.031]}$ for *line chart* (88.8% improvement), from $0.222_{[0.212,0.232]}$ to $0.026_{[0.024,0.028]}$ for *pie chart* (88.3% improvement), from $0.259_{[0.248,0.270]}$ to $0.034_{[0.031,0.037]}$ for *scatter plot* (86.9% improvement), and from $0.243_{[0.231,0.253]}$ to $0.035_{[0.032,0.037]}$ for *stream graph* (85.6% improvement).
- *Continuous colormaps.* Our method also outperforms the sequence-preserving method for both *heat map* and *choropleth map* ($t = -50.0, p < 0.0001$). Specifically, mean D_{dtw} is reduced from $0.236_{[0.226,0.246]}$ to $0.016_{[0.015,0.017]}$ for *heat map* (93.2% improvement), and from $0.286_{[0.265,0.305]}$ to $0.014_{[0.013,0.015]}$ for *choropleth map* (95.1% improvement).

More quantitative comparison results can be found in Supplementary Sec. 4.1.

5.3 Qualitative Examples

Synthetic visualizations. Fig. 9 presents five visualization images from the evaluation dataset, each paired with the ground-truth colormap (GT), colormap by palette-based [8] (Palette) or sequence-preserving [56] (Sequence), our CNN prediction (CNN), and the final colormap produced by our method (Final).

The left side presents a bar chart, line chart, and scatterplot, respectively. They are encoded with discrete colormaps shown in the GT row. Palette [8] shows two main deficiencies: first, the method by default extracts five colors, whilst ground-truth colormaps usually have more variants, e.g., six in the line chart and three in the scatterplot. Second, the method recovers color orders according to luminance channel, which often violates the color design rules. Moreover, the method often extracts a gray color that is actually not in the colormap. This is probably caused by the rendering process when drawing black grids and texts.

The fourth column presents a heatmap with grayscale colormap, which is not manageable by Sequence [56]. The method also produces incorrect colormaps for the US map in the last column, which is encoded with a rainbow colormap from the D3 library. Here, we can see that purple colors are not available anymore, which may be projected to red colors that exhibit similar hues with purple; and the blue colors are also omitted, probably because there are not many blue pixels in the map.

Real-world visualizations. Fig. 10 presents two more results of real-world visualizations. Corresponding colormap predictions by our method are presented underneath. Note here the visualizations are unseen to our neural network model. The left side presents a simple case of 5 single-hue discrete colors. Both Palette [8] method and ours can correctly extract the colormaps. The right side presents a more complicated example from [36]. Our method extracts a colormap that is very close to the ground truth. Yet, we can notice some differences: (i) The prediction does not include white color; this may be because the background filtering wrongly regards all white pixels as background; and (ii) the prediction generates dark red, rather than the red color as

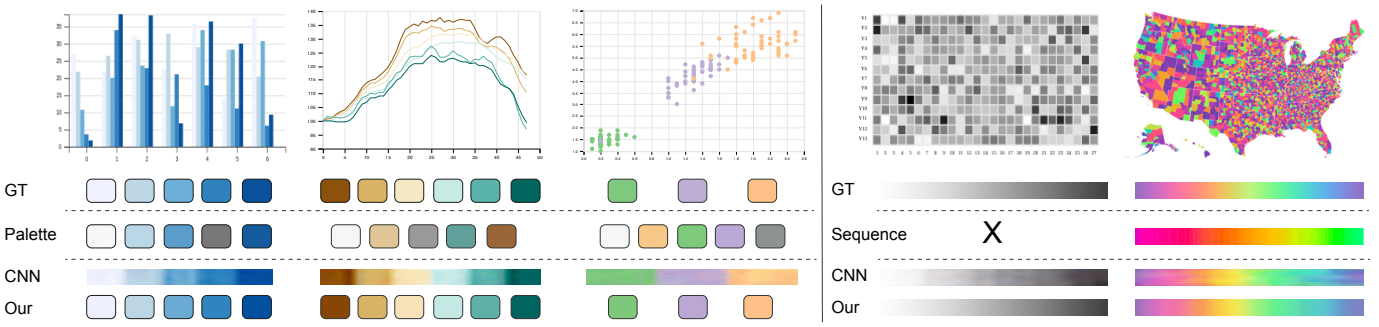


Fig. 9. Example colormap results by our approach *vs.* prior works. From top to bottom: input visualizations, ground-truth, colormaps extracted by palette-based [8] or sequence-preserving [56], our CNN predictions, and the final results produced by our method.

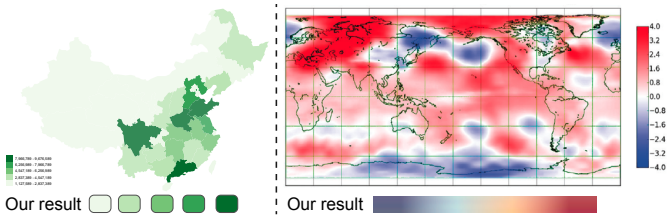


Fig. 10. Unseen visualizations and results by our method.

in the ground truth. Through a deep probe into the training data, we notice that the ground truth colormap is not in the corpus, and the network already tries its best to predict the most similar one. More real-world examples can be found in Supplementary Fig. 9 and Fig. 10.

5.4 Result Analysis

The experiment results show a big difference between synthetic and real-world visualizations by our approach, whilst those by palette-based and sequence-preserving methods remain almost the same. This is because our deep-learning-based approach is data-driven, of which the performance is affected by the difference between training and testing data distributions. Our model is trained on synthetic visualizations of limited combinations of chart types, data distributions, and colormaps, whilst real-world visualizations feature new chart types, different data distributions, and unseen colormaps. There is a dataset shift problem for real-world visualizations, yielding higher D_{dtw} than synthetic visualizations. On the other hand, palette-based and sequence-preserving methods are heuristic-based approaches, whose performances are highly dependent on hyperparameter tuning. Nevertheless, it is challenging to find optimal hyperparameters for diverse visualizations, causing less accurate predictions than our approach.

We also notice that the performance of prior methods, especially Sequence [56], are unstable and error-prone. We have striven to achieve reliable results for them by carefully adjusting their parameter settings. Nevertheless, our reimplementation may still bring in certain errors. Without considering these errors, we can reasonably come to the following conclusions by probing deep into the quantitative results and representative examples:

- *Discrete colormaps.* Palette [8] produces high distance $D_F(C_{gt}, C_{out})$ mainly for two reasons: (i) the method employs a fixed $k = 5$ for k-means clustering algorithm, thus it always generates five colors no matter how many colors are included in ground truth colormaps, as revealed by the line chart and scatterplot in Fig. 9. (ii) After retrieving the colors, the method sorts all the colors according to their luminance. This simple heuristic could easily generate wrong color orders, as in Fig. 9.
- *Continuous colormaps.* Sequence [56] often wrongly group colors with similar hues, or omits colors with low histograms. These heuristics can greatly affect the performance, in case of non-uniformly-distributed data; see the last example in Fig. 9 for example. Thanks again to the CNN model, our method is more robust to these situations.

Our method can effectively address these problems by processing fine predictions generated by the CNN, instead of directly processing visualization images. From Fig. 9, we can observe that the intermediate predictions exhibit these properties: (i) clear boundaries between distinct colors for discrete colormaps, promoting easy adaption of DBSCAN clustering instead of fix-sized k-means; (ii) well-balanced color histograms for continuous colormaps, making it easier for performing dimension reduction [56]; and (iii) roughly ordered colors, which can be used as reference for recovering the color orders.

6 Applications

This section introduces two applications based on colormaps produced by our methods: color design transfer (Sec. 6.1) and color-coding adaptation (Sec. 6.2).

6.1 Color Design Transfer

Inspired by example-based design, we can transfer color design from an existing visualization, by applying the extracted colormap to another visualization. Here, we take two visualization images as input, one as *reference* and the other as *target*. Our method automatically extracts two sorted lists of colors, denoted as $C_{ref} := \{c^i_{ref}\}^m_{i=1}$ and $C_{tgt} := \{c^i_{tgt}\}^n_{i=1}$, respectively. Specifically, we constrain the number of colors in the target visualization to be less than that in the reference visualization, i.e., $n \leq m$. Upon these conditions, we simply transfer the first n colors from C_{ref} to the target visualization. To accomplish this, we replace pixels in the target visualization of color c^j_{tgt} to color c^j_{ref}, where $1 \leq j \leq n$.

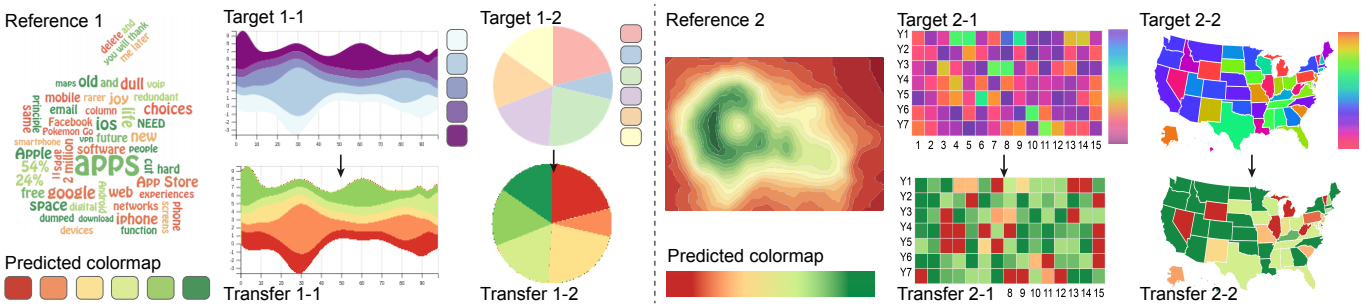


Fig. 11. Transferring colormaps from reference to target visualizations. Left reference: a word cloud [49] with a 6-class RdYlGn colormap. Right reference: a contour plot with a continuous RdYlGn colormap subject to the perceptual design principles.

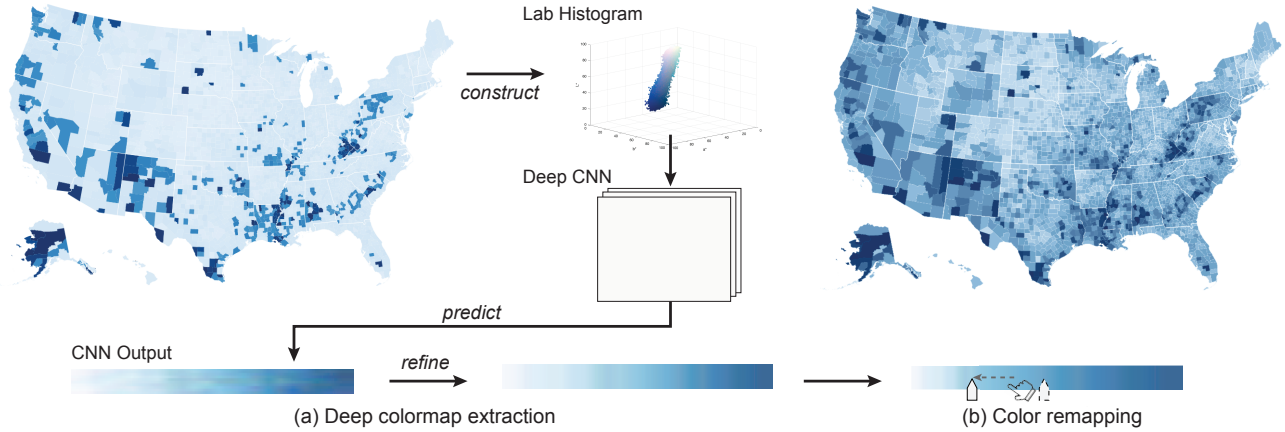


Fig. 12. From the input visualization image (without a given colormap) shown on top left, (a) we first construct a color histogram in Lab color space, then employ our trained deep convolutional neural network (CNN) to predict a colormap and refine the colormap; by then, (b) we can remap the predicted colormap to refine the visualization.

Fig. 11 shows some example results. The left one takes word cloud from a recent InfoVis paper [49] as the reference. It employs a conventional 6-class RdYlGn colormap from ColorBrewer and creates a pleasant visual appearance. Our method successfully extracts the colormap presented underneath. We transfer this colormap to (i) a ThemeRiver visualization with a 5-class BuPu colormap, which is difficult to distinguish light blue with white background; and (ii) a pie chart with a 6-class Pastel1 colormap, which also includes a light yellow color that is close to the white background. On the right side, a colored contour plot uses a continuous RdYlGn colormap. Our method again successfully extracts its colormap presented underneath. We then apply the colormap to refine a heatmap and choropleth map using rainbow colormaps that are not subject to human perception. After imitation, we can more easily compare the data values in the refined visualizations.

6.2 Color Remapping

Even with a well-chosen colormap, visually unappealing visualizations can still be created when inappropriate color ranges are applied to encode the unevenly-distributed data. In such scenarios, it is hard to perceive correct data values from a visualization. For instance, when a skewed dataset follows exponential or logarithmic distribution while a linear colormap is employed, only a small range of colors will be shown in the visualization. Tominski et al. [46] proposed to remap the color-coding in alignment with data distribution. Inspired by them, we develop a customized interface that allows for color remapping.

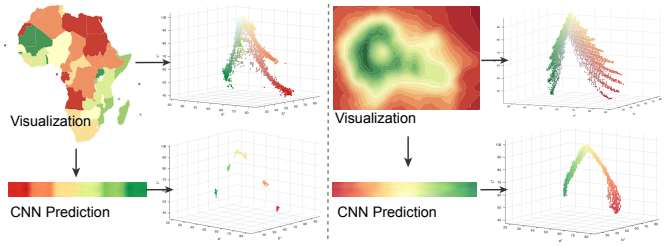


Fig. 13. Color histograms constructed from the CNN predictions exhibit much distinctive difference between continuous and discrete colormaps than from the input visualizations.

This application is designed for continuous colormaps. Specifically, we implement an intuitive *median-color* slider beneath the extracted colormap; see Fig. 12(b). Users can drag the slider to left/right to adjust the color-coding. An algorithm is developed to automatically resample the colors along the sequential color graph [56] based on the adjusted slider position. Fig. 12 illustrates a working example of the application. Here, a linear *BrBG* continuous colormap is employed to encode exponentially-distributed demographic data. Inappropriate mapping between data values and color ranges makes the choropleth map filled with brown colors. By dragging the *median-color* slider towards left, a better visualization (Fig. 12(b)) is generated.

7 Discussion

7.1 Deep Learning vs. Heuristic Approaches

The basis of our approach is a deep CNN model that learns to predict colormaps from input visualizations. The

prediction can be regarded as a mapping process from highly complex 3D histograms of original visualizations to relatively simple 2D colormaps. By this, our approach bypasses cumbersome parameter settings adopted in prior heuristic approaches, *e.g.*, thinning parameters in [56], and color distance threshold in [8]. In comparison with input visualizations, the predicted colormaps exhibit several distinctive benefits. First, the predictions provide much evident distinction between continuous and discrete colormaps. Fig. 13 presents a comparison of color histograms constructed directly from input visualizations and from corresponding predictions. Both visualizations employ RdYlGn colormaps: discrete for the left, and continuous for the right. However, color histograms constructed directly from the visualizations look similar to each other. In comparison, we can easily observe the differences between color histograms constructed from the predictions. Second, recovering color ordering is crucial for visualization, yet the task is very challenging. Many prior works employ simple heuristics based on hand-crafting the features to sort the colors, e.g., luminance values [8]. This can easily cause inappropriate color ordering; see Fig. 9 for examples. On the other hand, CNNs learn the features to make the predictions, which naturally contain the color ordering information.

7.2 Lessons Learned

We observe that the CNN model produces more accurate predictions for colormaps with a small number of colors (less than seven), and with more distinctive colors (e.g., multiple-hue categorical colormaps); see Supplementary Fig. 8 for details. Taking Fig. 9 as an example, CNN predictions for the leftmost bar chart (with five-class single-hue colormap) is relatively less accurate than the second line chart (with six-class diverging colormap) and the third scatterplot (with three-class multi-hue colormap). Nevertheless, single-hue discrete colormaps with too many colors are not encouraged, as they can impose difficulty for humans to perceive.

Although the network is trained on synthetic visualizations of fixed resolution 512×256, the CNN model is not constrained by the input image size. This is because our CNN model consumes a Lab color histogram as input, instead of directly consuming the input visualization image. Actually, our approach is not constrained to chart types and image sources as well. For instance, both the word cloud and contour plot used in Sec. 6.1 are unseen by the CNN model, and yet, our method can still extract their colormaps.

Besides the model trained with Lab color histograms, we also experimented with CNN models trained with the original visualization images and HSV color histograms as inputs. As depicted by Fig. 3, the image model can not preserve geometric invariance, which is regarded as a core requirement for deep neural networks for visualization [17]. In contrast, both the Lab and HSV models fulfill the requirement. On the other hand, Lab and HSV models achieve very similar training performances; see Supplementary Fig. 4 for details. We envision that a combination of both color spaces can enrich the feature map for a CNN to learn, which may lead to even better colormap prediction, as in the case of image classification [15].

7.3 Limitations

Though comparably effective and robust, our approach also exhibits several limitations.

First, our approach is data-driven, meaning that the network learns to predict based on the training dataset. This means that if there are new input patterns that the network has not seen before, it may produce poor results. Hence, we have striven to synthesize a dataset that covers a wide variety of visualizations in different styles by carefully considering data distributions, chart types, and colormaps. Yet, the space for visual design is too vast to be entirely covered. For instance, our synthetic dataset does not include tree-structured colors [45] and semantically meaningful colormaps [27].

Second, our approach works on color histograms rather than original images, limiting the scope of our work to linear colormaps only. Nevertheless, this is a common limitation for all existing colormap extraction methods that work on a color space [56], as no algorithm can explicitly distinguish if the histogram imbalance is caused by the underlying data distribution or by a non-linear colormap. We design an interactive interface that allows users to create nonlinear colormaps (Sec. 6.2).

Third, colormaps retrieved by our approach only indicate relative color ordering without explicit data semantics. That is, we can only tell if a color is ahead of or behind another one, but not able to infer what data value it represents. This limitation constrains our approach in applications based on both data and color mapping, such as interactive overlay [36]. Again, this is a common limitation also for existing colormap extraction methods. A complementary solution is to detect and recognize text labels in original images, and associate them with the retrieved colormaps [36].

8 Conclusion and Future Work

In this paper, we have presented a new approach to automatically extract colormaps from visualizations. The core of our method is a deep neural network that learns the mapping from Lab histograms to colormaps. To promote effective learning, we have i) synthesized a new dataset of diverse visualizations that covers a wide range of data distributions, chart types, and colormaps; ii) employed several feasible adaptions to the neural network, including conversion of 3D Lab histograms to 2D maps to fit with the CNN architectures, ASPP modules to create more nonlocal features, and fixed-size output image to handle conflict of limited neurons *vs.* enormous colors. Comparisons with state-of-the-art colormap extraction methods, sequence-preserving [56] and palette-based [8], also confirmed the advantages of our method; see both the quantitative evaluations and several representative examples. In the end, we presented two applications that can benefit from our method, color design transfer and color remapping.

Our work opens several directions for future research. First, our approach allows fully-automatic extraction of colormaps, enabling us to feasibly process vast amounts of visualization images. Inspired by TreeVis.net [39] and VIStory [58], we plan to exploit publications in visualization conferences and journals, and create a reference website

that summarizes color usage by the community. Second, a great challenge encountered in this work is the lack of proper training data. We will release the dataset to benefit future researches. Last, our network learns distinct features among color histograms by consuming vast amount of visualizations and corresponding colormaps. We would like to further enrich the training dataset by considering tree-structured colormaps [45], semantic colormaps [27], and user-created continuous colormaps [33], and add some noise to the synthetic visualizations to better cope with real-world visualizations. Nevertheless, a more appropriate approach would be to directly learn color design *rules*. It would be interesting to try using a graph neural network [55].

ACKNOWLEDGMENT

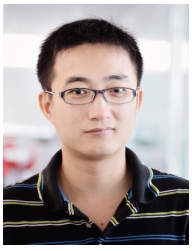
The authors wish to thank the anonymous reviewers for their valuable comments. This work is supported in part by National Natural Science Foundation of China (61802388) and SIAT Innovation Program for Excellent Young Researchers.

REFERENCES

[1] Colorbrewer2. http://colorbrewer2.org/.
[2] Colorcet. https://github.com/pyviz/colorcet.
[3] D3-scale-chromatic. https://github.com/d3/d3-scale-chromatic.
[4] J. Bernard, M. Steiger, S. Mittelstädt, S. Thum, D. Keim, and J. Kohlhammer. A survey and task-based quality assessment of static 2D colormaps. In *Proc. SPIE*, pages 9397:1–16, 2015.
[5] M. A. Borkin, A. A. Vo, Z. Bylinskii, P. Isola, S. Sunkavalli, A. Oliva, and H. Pfister. What makes a visualization memorable? *IEEE TVCG*, 19(12):2306–2315, 2013.
[6] D. Borland and R. M. T. Ii. Rainbow color map (still) considered harmful. *IEEE CG&A*, 27(2):14–17, 2007.
[7] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in Information visualization: using vision to think*. Morgan Kaufmann, 1999.
[8] H. Chang, O. Fried, Y. Liu, S. DiVerdi, and A. Finkelstein. Palette-based photo recoloring. *ACM Trans. Graph.*, 34(4):1–11, 2015.
[9] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE TPAMI*, 40(4):834–848, 2018.
[10] M. Chen, D. Ebert, H. Hagen, R. S. Laramee, R. van Liere, K. L. Ma, W. Ribarsky, G. Scheuermann, and D. Silver. Data, information, and knowledge in visualization. *IEEE CG&A*, 29(1):12–19, 2009.
[11] X. Chen, W. Zeng, Y. Lin, H. M. Al-maneea, J. Roberts, and R. Chang. Composition and configuration patterns in multiple-view visualizations. *IEEE TVCG*, 27(2):1514–1524, 2021.
[12] Z. Chen, W. Zeng, Z. Yang, L. Yu, C.-W. Fu, and H. Qu. LassoNet: Deep lasso-selection of 3D point clouds. *IEEE TVCG*, 26(1):195–204, 2020.
[13] S. Cheng, W. Xu, and K. Mueller. ColormapND: A data-driven approach and tool for mapping multivariate data to color. *IEEE TVCG*, 25(2):1361–1377, 2019.
[14] T. Giorgino. Computing and visualizing dynamic time warping alignments in R: the dtw package. *Journal of Statistical Software*, 31(7):1–24, 2009.
[15] S. N. Gowda and C. Yuan. Colornet: Investigating the importance of color spaces for image classification. *CoRR*, abs/1902.00267, 2019.
[16] C. C. Gramazio, D. H. Laidlaw, and K. B. Schloss. Colorgorical: Creating discriminable and preferable color palettes for information visualization. *IEEE TVCG*, 23(1):521–530, 2017.
[17] D. Haehn, J. Tompkin, and H. Pfister. Evaluating 'graphical perception' with CNNs. *IEEE TVCG*, 25(1):641–650, 2019.
[18] J. Harper and M. Agrawala. Converting basic d3 charts into reusable style templates. *IEEE TVCG*, 24(3):1274–1286, 2018.
[19] M. Harrower and C. A. Brewer. Colorbrewer.org: An online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27 – 37, 2003.
[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. CVPR*, pages 770–778, 2016.
[21] C. G. Healey. Choosing effective colours for data visualization. In *Proc. IEEE VIS*, pages 263–270, 1996.
[22] F. M. Hohman, M. Kahng, R. Pienta, and D. H. Chau. Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers. *IEEE TVCG*, 25(8):2674 – 2693, 2019.
[23] E. Hoque and M. Agrawala. Searching the visual style and structure of d3 visualizations. *IEEE TVCG*, 26(1):1236–1245, 2020.
[24] R. Irony, D. Cohen-Or, and D. Lischinski. Colorization by example. In *Proc. EGSR*, pages 201–210, 2005.
[25] D. Jung, W. Kim, H. Song, J.-i. Hwang, B. Lee, B. Kim, and J. Seo. Chartsense: Interactive data extraction from chart images. In *Proc. ACM CHI*, pages 6706–6717, 2017.
[26] S. Lee, M. Sips, and H. Seidel. Perceptually driven visibility optimization for categorical data visualization. *IEEE TVCG*, 19(10):1746–1757, 2013.
[27] S. Lin, J. Fortuna, C. Kulkarni, M. Stone, and J. Heer. Selecting semantically-resonant colors for data visualization. *Comput. Graph. Forum*, 32(3pt4):401–410, 2013.
[28] S. Liu, X. Wang, M. Liu, and J. Zhu. Towards Better Analysis of Machine Learning Models: A Visual Analytics Perspective. *Visual Informatics*, 1(1):48–56, 2017.
[29] Y. Ma, A. K. Tung, W. Wang, X. Gao, Z. Pan, and W. Chen. Scatternet: A deep subjective similarity model for visual analysis of scatterplots. *IEEE TVCG*, 26(3):1562–1576, 2018.
[30] S. Mittelstädt. *Methods for Effective Color Encoding and the Compensation of Contrast Effects*. Thesis, 2015.
[31] S. Mittelstädt, A. Stoffel, and D. A. Keim. Methods for compensating contrast effects in information visualization. *Comput. Graph. Forum*, 33(3):231–240, 2014.
[32] T. Munzner. *Visualization Analysis and Design*. CRC Press, 2014.
[33] P. Nardini, M. Chen, F. Samsel, R. Bujack, M. Böttinger, and G. Scheuermann. The making of continuous colormaps. *IEEE TVCG*, 2019.
[34] J. Pan, W. Chen, X. Zhao, S. Zhou, W. Zeng, M. Zhu, J. Chen, S. Fu, and Y. Wu. Exemplar-based layout fine-tuning for node-link diagrams. *IEEE TVCG*, 27(2):1655–1665, 2021.
[35] J. Poco and J. Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. *Comput. Graph. Forum*, 36(3):353–363, 2017.
[36] J. Poco, A. Mayhua, and J. Heer. Extracting and retargeting color mappings from bitmap images of visualizations. *IEEE TVCG*, 24(1):637–646, 2018.
[37] P. K. Robertson and J. F. O. Callaghan. The generation of color sequences for univariate and bivariate mapping. *IEEE CG&A*, 6(2):24–32, 1986.
[38] M. Savva, N. Kong, A. Chhajta, L. Fei-Fei, M. Agrawala, and J. Heer. Revision: automated classification, analysis and redesign of chart images. In *Proc. ACM UIST*, pages 393–402, 2011.
[39] H. J. Schulz. Treevis.net: A tree visualization reference. *IEEE CG&A*, 31(6):11–15, 2011.
[40] V. Setlur and M. C. Stone. A linguistic approach to categorical color assignment for data visualization. *IEEE TVCG*, 22(1):698–707, 2016.
[41] S. Silva, B. Sousa Santos, and J. Madeira. Using color in visualization: A survey. *Computers & Graphics*, 35(2):320–333, 2011.
[42] J. Tan, J. Echevarria, and Y. Gingold. Efficient palette-based decomposition and recoloring of images via rgbxy-space geometry. *ACM Trans. Graph.*, 37(6):262: 1–10, 2018.
[43] J. Tan, J.-M. Lien, and Y. Gingold. Decomposing images into layers via rgb-space geometry. *ACM Trans. Graph.*, 36(1):7: 1–10, 2017.
[44] T. Tang, R. Li, X. Wu, S. Liu, J. Knittel, S. Koch, L. Yu, P. Ren, T. Ertl, and Y. Wu. Plotthread: Creating expressive storyline visualizations using reinforcement learning. *IEEE TVCG*, 27(2):294–303, 2020.
[45] M. Tennekes and E. de Jonge. Tree colors: Color schemes for tree-structured data. *IEEE TVCG*, 20(12):2072–2081, 2014.
[46] C. Tominski, G. Fuchs, and H. Schumann. Task-driven color coding. In *Proc. Intl. Conf. Info. Vis.*, pages 373–380, 2008.
[47] Q. Wang, Z. Chen, Y. Wang, and H. Qu. Applying machine learning advances to data visualization: A survey on ML4VIS. *arXiv preprint arXiv:2012.00467*, 2020.
[48] Y. Wang, X. Chen, T. Ge, C. Bao, M. Sedlmair, C.-W. Fu, O. Deussen, and B. Chen. Optimizing color assignment for perception of class separability in multiclass scatterplots. *IEEE TVCG*, 25(1):820–829, 2019.

[49] Y. Wang, X. Chu, C. Bao, L. Zhu, O. Deussen, B. Chen, and M. Sedlmair. EdWordle: Consistency-preserving word cloud editing. *IEEE TVCG*, 24(1):647–656, 2018.
[50] C. Ware. Color sequences for univariate maps: Theory, experiments and principles. *IEEE CG&A*, 8(5):41–49, 1988.
[51] C. Ware. *Visual thinking: For design*. Elsevier, 2010.
[52] T. Welsh, M. Ashikhmin, and K. Mueller. Transferring color to greyscale images. *ACM Trans. Graph.*, 21(3):277–280, 2002.
[53] A. Wu, Y. Wang, X. Shu, D. Moritz, W. Cui, H. Zhang, D. Zhang, and H. Qu. Survey on artificial intelligence approaches for visualization data. *arXiv preprint arXiv:2102.01330*, 2021.
[54] A. Wu, L. Xie, B. Lee, Y. Wang, W. Cui, and H. Qu. Learning to automate chart layout configurations using crowdsourced paired comparison. In *Proc. ACM CHI*, 2021.
[55] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.
[56] M. J. Yoo, I. K. Lee, and S. Lee. Color sequence preserving decolorization. *Comput. Graph. Forum*, 34(2):373–383, 2015.
[57] L.-P. Yuan, Z. Zhou, J. Zhao, Y. Guo, F. Du, and H. Qu. InfoColorizer: Interactive recommendation of color palettes for infographics. *arXiv preprint arXiv:2102.02041*, 2021.
[58] W. Zeng, A. Dong, X. Chen, and Z.-l. Cheng. VIStory: interactive storyboard for exploring visual information in scientific publications. *Journal of Visualization*, 24(1):69–84, 2021.
[59] Q. Zhang, C. Xiao, H. Sun, and F. Tang. Palette-based image recoloring using color decomposition optimization. *IEEE TIP*, 26(4):1952–1964, 2017.
[60] J. Zhao, M. Fan, and M. Feng. Chartseer: Interactive steering exploratory visual analysis with machine intelligence. *IEEE TVCG*, 2021.
[61] L. Zhou and C. D. Hansen. A survey of colormaps in visualization. *IEEE TVCG*, 22(8):2051–2069, 2016.

Lin-Ping Yuan is currently a Ph.D. student in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology (HKUST). She obtained her B.Eng. degree in Software Engineering from Xi'an Jiaotong University, China in 2019. Her research interests include information visualization, visual analytics, and image processing.

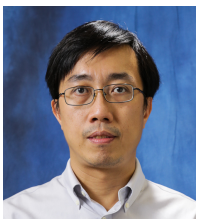


Wei Zeng is currently an associate professor at Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. He received his bachelor's degree (2011) and Ph.D. (2015), both in computer science, from Nanyang Technological University. He served as program committee members in various research conferences, including IEEE VIS, ChinaVis, PacificVis Poster, IVAPP. His research interests include all aspects of data visualization, with a particular focus on interactive techniques for urban data analysis, and data augmented design.

Siwei Fu is an associate research scientist in Zhejiang Lab. He received his Ph.D. degree in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. His main research interests are in visualization and human-computer interaction, with focuses on visual text analytics, multidimensional data visualization, and visualization recommendation.

Zhiliang Zeng received the B.S. degree from the Beijing Normal University, Zhuhai, and the M.Sc. degree in Computer Science and Engineering from the Chinese University of Hong Kong. He is currently a PhD student in Computer Science and Engineering of Chinese University of Hong Kong. His research interests include deep neural network and image segmentation, using neural network for indoor/outdoor scene analysis and application.

Haotian Li Haotian Li is currently a PhD student in Computer Science and Engineering at the Hong Kong University of Science and Technology (HKUST). His main research interests are data visualization, visual analytics and data mining, with emphasis on online learning and fintech. He received his BEng in Computer Engineering from HKUST. For more details, please refer to https://haotian-li.com/.



Chi-Wing Fu is currently an associate professor in the Chinese University of Hong Kong. He served as the co-chair of SIGGRAPH ASIA 2016's Technical Brief and Poster program, associate editor of IEEE Computer Graphics & Applications and Computer Graphics Forum, panel member in SIGGRAPH 2019 Doctoral Consortium, and program committee members in various research conferences, including SIGGRAPH Asia Technical Brief, SIGGRAPH Asia Emerging tech., IEEE visualization, CVPR, IEEE VR, VRST, Pacific Graphics, GMP, etc. His recent research interests include computation fabrication, point cloud processing, 3D computer vision, user interaction, and data visualization.

Huamin Qu is a professor in the Department of Computer Science and Engineering (CSE) at the Hong Kong University of Science and Technology (HKUST) and also the director of the interdisciplinary program office (IPO) of HKUST. He obtained a BS in Mathematics from Xi'an Jiaotong University, China, an MS and a PhD in Computer Science from the Stony Brook University. His main research interests are in visualization and human-computer interaction, with focuses on urban informatics, social network analysis, E-learning, text visualization, and explainable artificial intelligence (XAI).