Towards Automated Infographic Design: Deep Learning-based Auto-Generation of Extensible Timeline

Category: Research

Paper Type: algorithm/technique

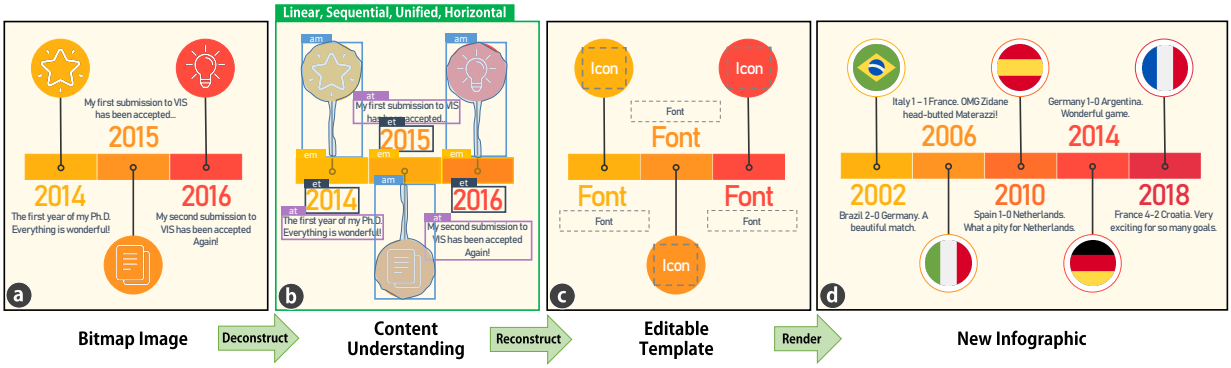

Fig. 1: An automated approach to extract an extensible timeline template from a bitmap image. a) Original bitmap image; b) Content understanding including global and local information of the timeline; c) Extensible template contains editable elements and their semantic roles; d) New timeline automatically generated with updated data.

Abstract—Designers need to consider not only perceptual effectiveness but also visual styles when creating an infographic. This process can be difficult and time consuming for professional designers, not to mention non-expert users, leading to the demands of *automated infographics design*. As a first step, we focus on timeline infographics, which have been widely used for centuries. We contribute an end-to-end approach that automatically extracts an extensible and extendable timeline template from a bitmap image. Our approach adopts a deconstruction and reconstruction paradigm. At the deconstruction stage, we propose a multi-task deep neural network that simultaneously parses two kinds of information from a bitmap timeline: 1) the global information, which includes the *representation*, *scale*, *layout*, and *orientation* of the timeline, and 2) the local information, which includes the *location*, *category*, and *pixels* of each visual element on the timeline. At the reconstruction stage, we propose a pipeline with three techniques, *i.e.*, *Non-Maximum Merging*, *Redundancy Recover*, and *DL GrabCut*, to extract an extensible template from the infographic, by utilizing the deconstruction results. To evaluate the effectiveness of our approach, we synthesize a timeline dataset (4296 images) and collect a real-world timeline dataset (393 images) from the Internet. We first report quantitative evaluation results of our approach over the two datasets. Then, we present examples of automatically extracted templates and timelines automatically generated based on these templates to qualitatively demonstrate the performance. The results confirm that our approach can effectively extract extensible templates from real-world timeline infographics.

Index Terms—Automated Infographic Design, Deep Learning-based Approach, Timeline Infographics, Multi-task Model

1 Introduction

Graphic designers have been producing infographics in a variety of fields, such as advertisement, business presentation, and journalism. Infographics have become widely used because of its effectiveness in spreading information [23, 54]. To inform data context and engage audience, infographics are often embellished with icons, shapes, and images in various styles [9, 30]. However, creating infographics is demanding. Designers should take into account not only perceptual effectiveness but also aesthetics, memorability, and engagement [9, 22, 23]. Some researchers have introduced design tools [8, 30, 54, 57] to alleviate the burden of creating infographics. However, these authoring tools require users to manually create infographics *from scratch*. The process remains difficult and time-consuming, especially for non-expert users, leading to the demands for *automated infographic design*.

Using template is an effective approach to enable automated infographic design, which has been widely used in commercial software, such as Microsoft PowerPoint and Adobe Illustrator. These systems can automatically generate infographics by plugging in data to a design template. Although easy to use, these systems typically only provide limited types of templates with default styles, which leads to a lack of diversity. By contrast, many infographics "in the wild" with diverse styles can only be accessed as images in a bitmap format. If users want to follow the styles of these bitmap infographics, they have to manually create their own infographics, which is difficult and tedious.

In this work, we investigate the methods of automatically extracting an extensible template from a bitmap infographic. Compared to *editable* templates, *extensible* templates not only contain the editable elements, but also the semantic roles of these elements, which enable automatic extension with updated data. Previous works [40, 41] attempt to extract visual encodings and color mappings from chart images on the basis of rules and machine learning (ML) methods, by utilizing the legends, axes, plot areas, and common layouts. However, the content of infographics can be unstructured and manifold. This makes it challenging to analyze infographic images and extract extensible templates from them. As a first step towards automated infographic design, we focus on the timeline infographics, which have been widely used for centuries and whose design space has been extensively studied [10].

Automatically extracting an extensible template from a bitmap timeline infographic is non-trivial. Particularly, two obstacles stand in the way of automatically extracting the extensible template from a timeline infographic image. First, it is challenging to automatically interpret a bitmap timeline infographic. Understanding the content of the infographic is necessary to automate the extraction. However, the elements in the infographic can be distributed in any place with any styles (*e.g.*, shapes, colors, and sizes *etc.*) This makes it difficult for a machine to

interpret the infographic which can only be accessed in pixels. Second, it is intricate to automatically convert a bitmap timeline infographic to be extensible. Even with an understanding of the infographic, we still cannot use a bitmap timeline infographic as a template. Even if the machine has already obtained structural information of the timeline, such as its type, orientation, and categories and locations of its elements, how to convert the timeline to an extensible template remains unclear, not to mention the information could be incorrect.

To address these challenges, we propose a novel end-to-end approach for automatically extracting an extensible template from a bitmap timeline infographic. Our approach adopts a deconstruction and reconstruction paradigm. We address the first challenge at the deconstruction stage. We propose a multi-task deep neural network (DNN) that simultaneously parses two kinds of information from a timeline image: global and local information. Global information includes the *representation*, *scale*, *layout*, and *orientation* of the timeline. Local information includes the *location*, *category*, and *pixels* of each visual element on the timeline. These two kinds of information provide a panorama of the timeline. We tackle the second challenge at the reconstruction stage. By utilizing the deconstruction results, we propose a pipeline with three techniques, *i.e.*, *Non-Maximum Merging*, *Redundancy Recover*, and *DL GrabCut*, to extract an extensible template from the infographic. The output can be used to generate new timelines with updated data.

To evaluate our approach, we synthesize a timeline dataset with 4296 labeled images and collect a real-world timeline dataset from Google Image [4], Pinterest [5], and FreePick [3]. We report quantitative evaluations of the two stages over the two datasets. We then present a set of examples with various visual styles, which are generated on the basis of automatically extracted templates, to qualitatively demonstrate performance. The results confirm that our approach can effectively extract an extensible template from real-world timeline infographics. Finally, we discuss lessons learned and future opportunities.

Our primary contribution is an automated approach to extracting extensible templates from bitmap infographic timelines. The approach consists of 1) a multi-task DNN that automatically deconstructs bitmap timeline infographics and 2) a pipeline that automatically reconstructs extensible templates. We evaluate our approach with quantitative evaluations and qualitatively demonstrate its effectiveness with examples.

2 Related Works

This section introduces prior studies that are most relevant to our work, including automated visualization design, computational interpretation of visualization, and deep learning-based object detection.

2.1 Automated Visualization Design

Automated visualization design systems aim at producing visual encodings for given input data based on both the criteria summarized by experts (*e.g.*, Bertin [7], Cleveland and McGill [13]) and constraints defined by users [39]. Prior work on automated visualization design can be classified into two general categories based on how the criteria are derived: rule-based and learning-based approaches.

Mackinlay's APT system [37] is a pioneering example that enumerates, filters, and ranks visualizations using expressiveness and perceptual effectiveness criteria. It was extended by SAGE [44], BOZ [52], and ShowMe [38] with additional considerations of data properties, low-level perceptual tasks, and candidate groupings. Recent systems like Voyager and Voyager 2 [55, 56] have further recommended data transformation (*e.g.*, normalization) in addition to visual encodings.

Foregoing explicit rules, researchers have recently designed learning-based systems that directly learn visualization designs from visualization corpora. DeepEye [36] applies ML models and design rules to determine whether a visualization is "good" or "bad" and recommends the "good" candidates. Data2Vis [15] uses a Recurrent Neural Network to automatically translate JSON-encoded datasets to Vega-lite [47] specifications. Draco [39] learns weights between hard and soft constraints that represent users' requirements and design guidelines. VizML [25] trains a fully-connected neural network to predict design choices based on input data. Although we also aim for automated design, these systems, however, cannot be adapted to infographics. They focus mainly on recommending *visual encodings* for the input data (*e.g.*, how to encode data using visual channels). By contrast, designing infographics requires additional attention to *visual styles* (*e.g.*, how to embellish the visualization with shapes and icons), which are omitted in these systems. In this regard, our work is inherently different from them.

2.2 Computational Interpretation of Visualization

Computational interpretation of visualization seeks to enable machines to understand the content of visualization images (*e.g.*, data, styles, and visual encodings) For this goal, researchers have proposed various methods, including semi-automatic *vs.* fully automatic and on vector *vs.* bitmap images. We focus on fully automatic methods on bitmap images. According to their targets, prior methods can be divided into two categories: for charts and for infographics.

A general pipeline when interpreting a chart is to first identify the type of the chart via classifications, then detect elements (*e.g.*, marks or text) in the chart, and finally extract the underlying information (*e.g.*, data or visual encodings). As a pioneer, Savva et al. introduced ReVision [48], in which the graphical and textual features are fed into a support vector machine (SVM) model for a chart type classification. ReVision then localizes the marks and extracts data from pie and bar charts by using a carefully designed multi-steps method based on image processing and heuristics. Siegel et al. [50] extended the method of ReVision to handle line charts. They developed a convolutional neural network (CNN) for the chart classification and designed a heuristic approach to use legend information for data extraction. Recently Kafle et al. [27] have used a deep dual-network model to directly parse the data from bar charts without heuristic rules. Instead of extracting the data of charts, Poco and Heer [40] aimed to recover the visual encodings. To complete the task successfully, they proposed a state-of-the-art approach to interpret the text in a multi-stage pipeline, which combines ML and heuristics methods. Building on this, Poco et al. [41] further explored the color mapping extraction of visualization images.

Apart from charts, researchers have explored the computational interpretation of infographics. Bylinskii et al. [12] used fully convolutional networks (FCNs) to predict the visual saliency of an infographic. Bylinskii et al. [11] also applied DNNs to automatically select representative textual and visual elements from an infographic. On the basis of several deep learning models, Kembhavi et al. [29] designed a multi-stage approach to parse the relationships among elements in diagrams in science textbooks. Although these methods enable computational understanding of an infographic from certain perspectives, the information they interpret cannot be used to reconstruct an extensible template (*e.g.*, how to change or extend the content of an infographic is unknown). We take a first step towards the interpretation of infographics for an automated design purpose. Unlike using multiple models and handcrafted features, our approach uses one end-to-end DNN to complete the interpretation.

2.3 Deep Learning-based Object Detection

To extracting an extensible template, we need to understand each object on it. We achieve this goal with deep learning-based object detection. Object detection is a computer vision task whose goal is to localize each object using a bounding box (*i.e.*, *where*) and classify its category (*i.e.*, *what*). Deep learning-based object detection methods can either be one-stage [34, 35, 42] or multi-stage [17, 18, 28, 43]. One-stage models directly predict objects' bounding box and category without involving intermediate tasks. YOLO [42] is a representative one-stage model that divides the image into small cells and predicts bounding boxes for each cell. One-stage models have the advantage of fast detection in real time, which affects accuracy. By contrast, multi-stage models can predict accurately, but are often less time efficient. Multi-stage models, such as RCNN [18], usually first propose a manageable number of candidate regions (*region proposals*) that may contain objects. If an object exists within, then they will predict its bounding box and category. Time consumption is not our first priority, so we base our work on a multi-stage model. Mask R-CNN [28] is a leading multi-stage model in several benchmarks. It can further predict the pixels of an object within its bounding box (*i.e.*, *Instance Segmentation*). We extend Mask R-CNN to interpret not only the information of objects (*i.e.*, local) but also that of the entire timeline infographic (*i.e.*, global). To the best of our knowledge, we are the first to adopt this kind of instance segmentation networks to deal with the infographics interpretation problem.

3 Problem Statement

This section introduces the background of timeline infographics and the problem, overview of the proposed approach, and the datasets.

3.1 Background

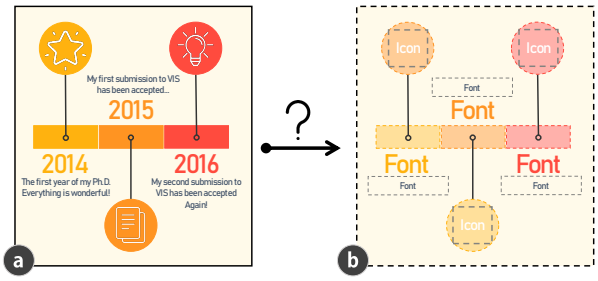

Fig. 2: Given a bitmap timeline infographic, we seek to automatically extract its extensible template.

Timeline infographics have been recently investigated by Brehmer et al. [10]. We briefly describe the background as follows:

- *Timeline Data.* A timeline presents interval event data (*i.e.*, a sequence of events), which is different from continuous quantitative time-series data. A timeline infographic for storytelling usually has a small underlying dataset because the storyteller is assumed to have already distilled the narrative points from the raw dataset.
- *Timeline Design.* A timeline can be described using three dimensions, namely, No more than five options are available for each dimension (Table 1). Besides, there are only 20 viable combinations of these options. Thus, the combinations can be used as the *type* of timelines. These dimensions indicate how the events are organized in a timeline. For example, events are placed along a straight line in a *linear* representation, which is the most common way to represent a timeline. Typically, an event is visually encoded by a graphical mark, such as the rectangles in Fig. 2. The position of this mark is used to encode the occurred time of the event. Extra annotations (*e.g.*, text or icons) are added, commonly adjacent to the event mark, to depicts the details of an event.

Table 1: The design dimensions to depict a timeline.

Desgin Options	
Representation	Linear, Radial, Grid, Spiral, Arbitrary
Scale	Chronological, Relative, Logarithmic, Sequential, Sequential + Interim Duration
Layout	Unified, Faceted, Segmented, Faceted + Segmented

In practice, infographic timelines are widely spread in the form of bitmap images. However, they are not easy to reproduce. Given a bitmap timeline, our goal is to automatically extract its extensible template to generate new timelines with updated data (Fig. 2),. To automatically extract an extensible template from an infographic image, two requirements should be fulfilled:

- **Parse the content.** The machine should first parse the content of the image. A computational understanding of an image can be represented as a structural information, which is necessary for an automation process. However, the infographic image can only be accessed in pixels, which is a 3D byte array with the shape of $width \times height \times RGB$. A process is required to take the bitmap image as input and output its structural information.
- **Construct the template.** With the structural information of the image as a basis, the machine should be able to automatically construct an extensible template out of it. The template should contain detail information (*e.g.*, position, color, font, and shape) of the elements to be reused and the elements to be updated. Given the image and its structural information, another process should be involved to extract such types of detail information.

3.2 Approach Overview

To fulfill the aforementioned two requirements, we design a two-step approach, starting from defining the input and output of each step.

Deconstruction. The goal of the first step (Fig. 1a and Fig. 1b) is to parse structural information from the input, a bitmap timeline infographic I. For the output, we define two kinds of information, namely, the global one G and the local one L. The global information is about the entire timeline, including its *representation*, *scale*, *layout*, and *orientation*. The local information is about each individual element, including its category (*what*), location (*where*), and the pixel-wise mask (*which pixels*). Therefore, the ideal process of the first step can be formulated as a mapping function f:

$$f : I \rightarrow (G, L) \tag{1}$$

We propose to approximate f using a DNN model $h \approx f$ with a set of parameters Θ. This set of parameters Θ can be learned from a corpus $\mathbb{C} = \{(I_i : (G_i, L_i))\}_{i=1}^{n}$, where each entry $(I_i : (G_i, L_i))$ is a bitmap image associated with its global and local information. Hence, we can obtain the output via $(G, L) = h(I|\Theta)$.

Reconstruction. To reconstruct the extensible template, a function g should take the bitmap infographics I and its global and local information G, L as the input, and return the detail information about elements to be reused E_r (*e.g.*, the rectangle and circle marks in Fig. 2a) and elements to be updated E_u (*e.g.*, the text and icons in Fig. 2a), *i.e.*,

$$g : (I, G, L) \rightarrow (E_r, E_u) \tag{2}$$

E is a set of elements, each of which is represented as a set of attributes, *i.e.*, $E = \{e^i := (a^1, a^2, ..., a^m)\}_{i=1}^{n}$. According to G and L, we can infer attributes of elements in E_r and E_u, such as *size*, *shape*, *color*, *position*, and *offset* to others. We highlight the necessary attributes for enabling extensible templates. For E_r, the essential attribute is the graphical marks to be reused (*e.g.*, the rectangle marks in Fig. 2a). Hence, we need to segment the pixels of E_r from the original image. As for E_u, the attributes related to *font* (*e.g.*, *font family*, *size*, *color*, *etc.*) must be identified to maintain the styles of the updated content. In addition, we note that the outputs from h may not be perfect, reducing the quality of the outputs of g. Thus, g should be smart enough to correct errors in G and L as much as possible.

Considering these issues, we design a heuristic-based pipeline, with three novel techniques, as g to automatically output E_r and E_u.

3.3 Datasets

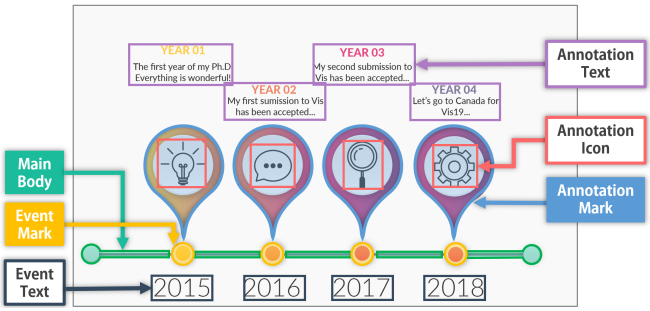

Fig. 3: Categories of elements in a timeline infographic. The *event mark*, *annotation mark*, and *main body* can be reused, while others need to be updated.

We use two datasets to train the model h and evaluate our approach. The first one (referred to as D_1) is a synthetic dataset. We extended TimelineStoryteller [10], a timeline authoring tool, to generate D_1, covering all types of timeline. The second dataset (referred to as D_2) consists of real-world timelines, collected from Google Image [4], Pinterest [5], and FreePicker [3] by using the search keywords *timeline infographics* and *infographic timeline*. D_2 has more diverse styles, especially for marks, and it covers most common types of timeline. The resolutions of images are in the range of $[512, 3880] \times [512, 4330]$. We make three assumptions for simplifications: First, we assume that a timeline has less than 20 events. Second, we assume all events in a timeline have the same number and types of annotations (*e.g.*, text and icon). Third, we exclude the titles, footnotes, and legends in timelines.

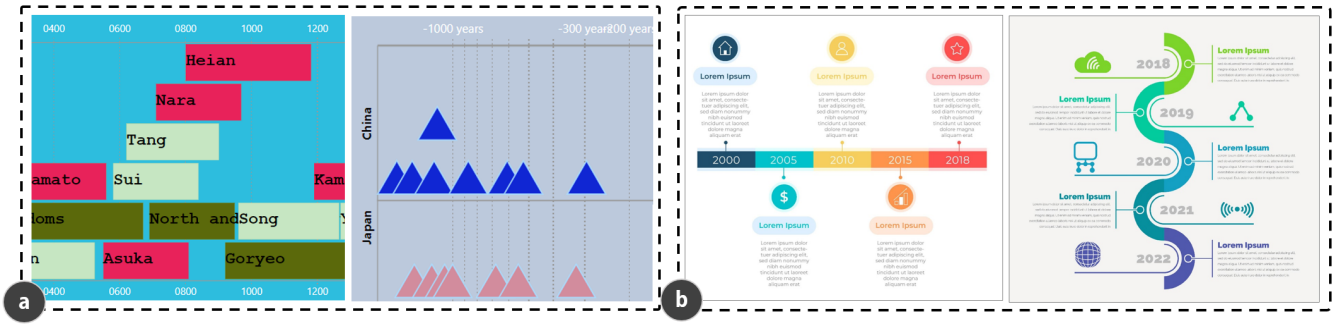

Fig. 4: Example timelines from: a) a synthetic dataset D_1, which shows two different representations, and b) a real-world dataset D_2, which shows two different orientations.

Table 2: The number of annotations per category of each dataset.

Dataset	#Event Mark	#Event Text	#Annot. Mark	#Annot. Text	#Annot. Icon	#Main Body
D_1	83498	61324	4030	60036	-	-
D_2	2318	2305	2227	2937	1497	1340

Collection. For D_1, the open source authoring tool allows us to generate timeline image with various visual encodings and styles. We generated timeline images using nine embedded datasets of the tool to cover the design space of timeline. To increase diversity, we randomly modified timeline orientation, the style of graphical marks (including color, size, and shape), texts (font, size, color, offset to others), and the background (color) in a curated range that guarantee the viability of the timeline. We created 9592 timelines in this process.

For D_2, we implemented crawlers to download the search results. The crawling process was manually monitored and stopped when 10 consecutive return results are not timelines. We collected 1138 timelines in this process. Following, four of the coauthors separately reviewed all the timelines to remove the repeated and problematic instances, such as images with heavy watermark or with very low resolutions (*i.e.*, smaller than 512×512). They came up with in 412 acceptable timelines. The scale of D_2 is consistent with manually collected visualization datasets in similar research [10,40,41]. Among the five representations summarized in [10], *radial*, *grid*, or *spiral* representations appear only 19/412 (4.6%) timelines, whereas the rest 393 timelines are with *linear* or *arbitrary* representations. This ratio is consistent with [10] (23/263, 8.7%). Considering the scarce number of the *radial*, *grid*, or *spiral* representations, we excluded them in D_1 and D_2 and focus on the more common *linear* and *arbitrary* representations.

Labeling. To identify the categories of elements in a timeline, four of the coauthors independently reviewed all the timelines in D_1 and D_2. Each of them iteratively summarized a set of mutually exclusive categories that can be used to depict elements in a timeline infographic. Gathering the reviews resulted in six categories (Fig. 3). We explain the details of these categories on our project page.[1]

Each timeline in D_1 was then converted from SVG to bitmap format and annotated with its representation, scale, layout, and orientation. We also analyzed the SVG and the bitmap to generate the annotations for each element in a timeline, including its category (from the label sets in Fig. 3), bounding box (referred to as *bbox*), and pixel-wise mask (referred to as *mask*). For each timeline in D_2, we manually annotated its representation, scale, layout, and orientation, as well as the category, bbox, and mask of each element, by using our annotation tool that is built on Microsoft PowerPoint. Finally, D_1 contains 4296 timelines, whereas D_2 contains 393. Figure 4 and Table 2 present samples and statistics of these timelines, respectively.

4 Deconstruction

Parsing bitmap timeline infographics and further extracting structure information from it are difficult because of the absence of fixed rules for timelines element styles and layouts. Our approach parses the timeline infographics from two perspectives, namely, global and local. In contrast with prior studies [40,50] that extracted similar information from charts using different methods in multiple steps, our study involves the use of a DNN to parse structure information in one shot.

4.1 Parsing Global Information

Our dataset comprises 10 types of timelines. The *type* depicts the high-level structures of a timeline, which are necessary for constructing an extensible template. In addition to the *type* of timeline, the *orientation*, which could be *horizontal*, *vertical*, and *others*, is equally indispensable for the template. As *type* and *orientation* only involve a few discrete choices, we can identify them through classification.

Taking into account that CNN models have shown excellent capability in chart classification [26, 40, 50], we propose a CNN-based classifier to recognize the *type* and *orientation* of a timeline. As shown in Fig. 5, a CNN-based classifier with two classification heads is designed to achieve the classifications of timeline *type* and *orientation* simultaneously, which reduces the training cost.

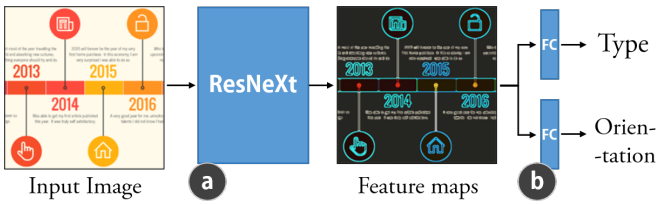

Fig. 5: Initial architecture to parse the global information. After extracting the feature maps of an image, two FC layers are used to classify its *type* and *orientation*.

Many CNN architectures have been proposed (*e.g.*, AlexNet [31], GoogLeNet [53]). Since ResNeXt [58] achieves a state-of-the-art performance in many computer vision tasks, we use ResNeXt (Fig. 5a), to extract the features of a timeline infographic. We then use two groups of fully connected (FC) layers (Fig. 5b) as Class heads to predict the timeline's *type* and *orientation* based on the feature maps.

4.2 Parsing Local Information

After parsing the global information, the machine should further extract the local information of the timeline. We have defined six categories of elements (Fig. 3) in a timeline. We need to detect each element in the timeline (where and what) and segment it from others (which pixels).

To tackle these tasks, a possible solution is to solve them one by one using well-established methods. For example, we can use sliding windows [32] to localize elements, then apply classifiers (*e.g.*, SVM) to determine the category of the element within, and lastly segment the element from the image. This multi-step solution can be effective and has been used in previous works [26, 29, 40, 50]. However, given the ad-hoc nature, extending this solution to other scenarios is challenging. Therefore, we prefer to adopt a unified method to complete all tasks.

Considering that we have already extracted the feature maps of the infographic in Sect. 4.1, we propose to reuse these feature maps, which contain rich information of the image. Specifically, we extend the classification models in Fig. 5 by adding components for object detection to parse the local information. We use Mask R-CNN [28], which is a leading architecture that can detect objects and predict the

[1] https://auto-infog-timeline.github.io

pixel mask for them, to finish the extension. As such, our model can simultaneously finish all five tasks (*i.e.*, two tasks in global level, three tasks in local level) in one shot. The complete architecture is depicted in Fig. 6. We successfully train this multi-task learning model and achieve a good performance.

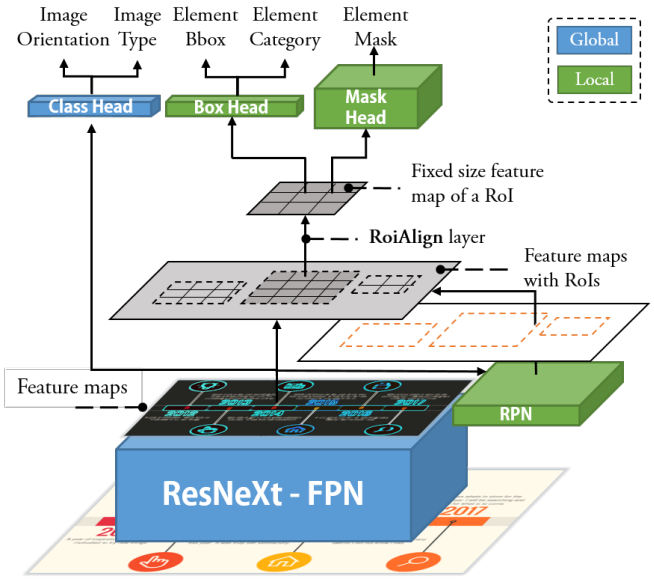

Fig. 6: Complete architecture to parse both global and local information simultaneously. Apart from the components in Fig. 5, we add more components (in green color) to parse local information.

In the architecture (Fig. 6), we first extend ResNeXt with Feature Pyramid Network [33] (FPN). FPN is a top-down architecture and can build semantically strong feature maps at multiple scales using the feature maps from ResNeXt. FPN makes our model scale-invariant and able to handle images of vastly different resolution. We then feed the feature maps from the ResNeXt-FPN into a Region Proposal Network [43] (RPN) to localize elements in a timeline. RPN is an FCN that simultaneously predicts element locations (*i.e.*, by bbox) and objectness scores (*i.e.*, whether there is an object within the bbox) in an image. These element location hypotheses are then be used to extract regions of interest (RoIs) from the feature maps. Each RoI is normalized to fixed size using a RoIAlign layer and then passed to two heads, namely, a Box head and a Mask head. The Box head uses two sibling FC layers to classify the category and regress the bbox of the element. The Mask head uses an FCN for predicting the pixels of the element within the bbox. The outputs of Box and Mask heads on a RoI is referred to as *DT* (*i.e.*, detection). Additional details on architecture and extension are presented on the project page.

4.3 Training

We set hyper-parameters by mostly following [28]. Our initial testing confirmed that these hyper-parameters work well for our model on infographics, although they are mainly chosen for natural images.

Fine-tuning on pre-trained networks. A large-scale dataset is required to train our model for a good performance. However, our two datasets are small and have only 4689 images in total. A key advantage of deep learning model is that the feature representations learned on a pre-trained task can transfer to another task [24]. This well-established strategy not only speeds up the convergence of training but also enable good performance on tasks with small datasets. Therefore, we initialize the model using weights pre-trained on ImageNet [14], which is a large dataset containing millions of images across 1000 classes. We then fine-tune the model on our datasets.

Multi-task loss. Our model is optimized for a multi-task loss function that consists of seven losses:

$$\begin{aligned}\mathcal{L} = {} & \lambda_1 \mathcal{L}_{Image_{type}} + \lambda_2 \mathcal{L}_{Image_{orientation}} \\ & + \lambda_3 \mathcal{L}_{RoI_{objectness}} + \lambda_4 \mathcal{L}_{RoI_{bbox}} \\ & + \lambda_5 \mathcal{L}_{DT_{type}} + \lambda_6 \mathcal{L}_{DT_{bbox}} + \lambda_7 \mathcal{L}_{DT_{mask}}\end{aligned} \tag{3}$$

Table 3: Multi-tasks loss functions of our model.

	Target	Type	Loss	Weight
$\mathcal{L}_{Image_{type}}$	Image	Classification	Cross-Entropy	0.15
$\mathcal{L}_{Image_{orientation}}$	Image	Classification	Cross-Entropy	0.15
$\mathcal{L}_{RoI_{objectness}}$	RoI	Classification	Cross-Entropy	1
$\mathcal{L}_{RoI_{bbox}}$	RoI	Regression	Smooth L_1	1
$\mathcal{L}_{DT_{type}}$	DT	Classification	Cross-Entropy	1
$\mathcal{L}_{DT_{bbox}}$	DT	Regression	Smooth L_1	1
$\mathcal{L}_{DT_{mask}}$	DT	Classification	Cross-Entropy	1

The summary of these losses is presented in Table 3. We briefly overview each loss and refer the reader to the project page for details:

- *Loss Functions on Images.* The $\mathcal{L}_{Image_{type}}$ and $\mathcal{L}_{Image_{orientation}}$ are defined on the entire image. They are computed using the output of classifications of timeline types and orientations, respectively. Both them are cross-entropy losses.
- *Loss Functions on RoIs.* The $\mathcal{L}_{RoI_{objectness}}$ and $\mathcal{L}_{RoI_{bbox}}$ are defined on each RoI. They are computed using the output of RPN. The first loss is a binary cross-entropy loss for the classification of whether the proposed region contains an object. The second loss is a $smooth_{L_1}$ loss defined in [18] for the bbox regression. It is only activated for RoIs that contain objects.
- *Loss Functions on DTs.* The remaining three losses are defined on each DT. $\mathcal{L}_{DT_{type}}$ and $\mathcal{L}_{DT_{bbox}}$ are computed using the outputs of Box head, while $\mathcal{L}_{DT_{mask}}$ is computed using those of Mask head. Similarly, $\mathcal{L}_{DT_{type}}$ is a cross-entropy loss for the classification over six pre-defined element categories and a "catch all" background. Both $\mathcal{L}_{DT_{bbox}}$ and $\mathcal{L}_{DT_{mask}}$ are only activated on foreground elements. $\mathcal{L}_{DT_{bbox}}$ is also a $smooth_{L_1}$ loss, which is similar to $\mathcal{L}_{RoI_{bbox}}$, for further refining the bbox outputted by RPN. To classify whether the pixels within the bbox belong to the object, $\mathcal{L}_{\text{mask}}$ is defined as the average binary cross-entropy loss over the pixels within the bbox.

The hyper-parameters λ in Equation 3 control the balance between these seven task losses. We note that the losses defined on the entire image (*i.e.*, $\mathcal{L}_{Image_{type}}$ and $\mathcal{L}_{Image_{orientation}}$) are not on the same scale with other losses (which are defined on the local regions of the image). Therefore, we empirically set a smaller λ to them (*i.e.*, 0.15) and follow previous works [28] to keep other losses as 1.

4.4 Validating

Our model is implemented using the Pytorch [6]. We implemented two types of CNN backbone for our model, namely, ResNeXt-50 (R50) and ResNeXt-101 (R101), following the standard configurations [58]. R50 has 50 layers, which is more lightweight and easier to train, while R101 has 101 layers, which performs better in computer vision tasks at the cost of efficiency and is more difficult to train. We trained these two implementations of our model using D_1 and D_2 together. We randomly split the images in D_1 and D_2 into $9:1$ such that no testing sample is in the training set (*i.e.*, these testing samples have never been trained). To increase the diversity of the training data, we conduct several data augmentation strategies, including random horizontal or vertical flip, random rotations, and random color channels swap. Finally, the number of training samples for one epoch is 33760. We evaluated models trained with 10 epochs on the two datasets separately. We first report the performance of parsing global information and then report the average precision (*AP*) on parsing local information. Reported numbers are averaged over 10 independent runs.

Parsing Global Information: To access the performance of our model on the two classification tasks (*i.e.*, 10 classes on timeline types and 3 classes on orientations), we calculate the precision, recall and F1-score. Table 4 presents the results.

For *type* and *orientation* classifications, both implementations achieve good performance on D_1 and D_2. As expected, the implementation with R101 has a better performance on D_1 and D_2 than that with R50. The classification of *type* on D_2 performs worse than that

Table 4: Classification of timeline types and orientations.

Dataset / Backbone	Type			Orientation		
	Pre.%	*Rec.%*	*F1%*	*Pre.%*	*Rec.%*	*F1%*
D_1 / R50	99.1	99.1	99.1	100.0	100.0	100.0
D_1 / R101	99.5	99.5	99.5	100.0	100.0	100.0
D_2 / R50	88.7	86.4	87.5	97.7	97.1	97.4
D_2 / R101	92.2	90.9	91.5	97.7	97.1	97.4

on D_1, which is largely due to the more diversity and small size of D_2. Nevertheless, the F1 score is still higher than 90% when using R101.

Parsing Local Information: To evaluate the performance on parsing local information, we use the metrics in COCO challenge [1]. COCO is a large-scale object detection and segmentation dataset that contains more than 330K images with high-quality annotations. It is a leading platform for evaluating object detection methods using *AP* metrics [16] that access the three tasks (*i.e.*, *what*, *where*, and *which pixels*) together. Basically, *AP* is a measure of precision-recall trade off calculated using all possible confidence level that is represented by the classification score associated with each predicted bbox. Intuitively, *AP* is the area under the precision-recall curve (Fig. 7). To calculate the precision and recall at a confidence level, we first need to calculate the intersection over union (IoU) between each predicted bbox B_p and its corresponding ground truth B_gt by $IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$. If the IoU exceeds a threshold (*e.g.*, 0.5), then the prediction is considered as a true positive, otherwise a false positive. We can then further calculate the precision and recall over all confidence level to draw the curve.

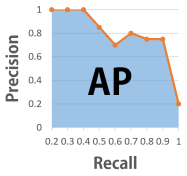

Fig. 7: AP.

Table 5: Average Precision of parsing local information.

Dataset / Backbone	BBox			Mask		
	$AP_{50:95}$	AP_{50}	AP_{75}	$AP_{50:95}$	AP_{50}	AP_{75}
D_1 / R50	79.0	93.6	88.0	79.8	96.4	91.6
D_1 / R101	81.9	93.9	89.1	79.9	96.9	91.1
D_2 / R50	53.4	79.3	61.8	56.9	80.1	61.6
D_2 / R101	56.4	81.7	64.9	59.1	82.5	65.1
*COCO**	39.8	62.3	43.4	37.1	60.0	39.4

*A state-of-the-art performance on COCO dataset reported by [28].

Table 5 presents the *AP* of our model on the two datasets. The higher the *AP*, the better it is. We provide a state-of-the-art performance on COCO reported by He et al. [28] as a background, due to the lack of benchmarks. $AP_{50:95}$ is the average *AP* over different *IoU*, from 0.5 to 95 with step 0.05. AP_{75} and AP_{50} is the *AP* calculated at $IoU = 0.75$ and $IoU = 0.5$, respectively. The larger the *IoU*, the stricter the metric will be. As indicated in Table 5, our model achieves high *AP* on bbox detection and pixel segmentation on D_1. This result is because the overall diversity of D_1 is limited, the size of D_1 is big enough in relation to its diversity, and the auto-generated annotations of D_1 are perfect enabling an effective learning of the model. By contrast, D_2 has a more diversity, smaller size, and imperfect annotations in comparison with D_1, leading to a decreasing in performance. Nevertheless, our model still achieves an acceptable performance on D_2, considering the state-of-the-art performance on COCO. We further discuss the perfectness of annotations of D_2 in Sect. 5.5.

5 Reconstruction

After understanding the content of a timeline infographic, the next problem is how to automatically reconstruct an extensible template from it. We introduce a reconstruction pipeline (Fig. 8) that exploits the outputs from the previous step to extract an extensible template.

Our pipeline first eliminates repeated bbox using *Non-Maximum Merging* (*NMM*) together with *Non-Maximum Suppression* (*NMS*), then infer the missing elements using *Redundancy Recovery* (*RR*). Next, we introduce *DL GrabCut* to automatically extract high-quality graphical marks for reuse. Finally, we identify the font of *event text* and *annotation text* use publicly available API. A quantitative validation and example results confirm the effectiveness of our pipeline.

5.1 Eliminate Repeated BBoxes: Non-Maximum Merging

Multiple predicted bboxes may exist on one object during object detection, such as the two bboxes in Fig. 9a. We need to remove these repeated bboxes. For natural images, one commonly used method is *NMS* [18]. *NMS* iteratively eliminates bbox whose confidence score (*i.e.*, the classification score) is less than a predefined threshold. For instance, in Fig. 9a, with a confidence threshold of 0.8, the pink bbox with 0.58 score will be eliminated after the *NMS* while the red one will be outputted. However, for infographics, a part of an object may still be a "complete" object, which hinders the effectiveness of *NMS*. For example, in Fig. 9b, the mark in the steel blue bbox and the part of it in the deep blue bbox are both valid annotation marks. In such case, each of the two bboxes will be assigned a high confidence score (*e.g.*, 1.0). Therefore the *NMS* cannot eliminate the repeated box.

Therefore, we design an *NMM* algorithm to eliminate repeated bboxes. Specifically, for elements with the same category, we rank them using the confidence score plus the area of bbox (normalize to $[0, 1]$) instead of only using the confidence score. For the top 1 element's bbox, we merge other elements' bboxes that overlap with it and exceed a *IoU* threshold to form the union bbox of them. This process is repeated until all overlapping boxes are merged. Fig. 9b shows the boxes before and after *NMM*.

Our pipeline supports both *NMS* and *NMM*. In practice, for repeated bboxes, we separately apply *NMS* and *NMM* and then check the consistency of the shapes between the resulted bboxes and other non-repeated bboxes with the same category. The most consistent results are kept.

5.2 Fix Failed Detections: Redundancy Recovery

One limitation of our model is that it may detect elements with wrong categories (*i.e.*, false positive) or miss elements (*i.e.*, false negative). To fix these failed defections, we leverage the redundant information of timelines (*e.g.*, each event has the same type of annotations). Specifically, for the elements in a timeline, we first group them along the timeline orientation into clusters, each of which represents an event. Then, for each event, we use the statistics of the elements grouped by events to verify it and attempt to fix the failed detections. This process is referred to as *RR* (Redundancy Recovery).

Incorrectly classified elements. Some elements in an infographic can be classified to incorrect categories. For example, a short annotation text with a fancy font can be incorrectly classified as an annotation icon. We adopt a voting mechanism to attempt to fix these incorrect classifications. For instance, if more than half of the events contain annotation text, then an annotation icon, whose bbox has the same shape with these annotation texts, of an event should be classified as an annotation text. Given an event can have multiple annotation texts, we restrict that only the annotation texts with the consistent shape of bbox can vote for each other. This rule is also applied to other categories.

Missing elements. We also use a similar voting mechanism to infer the undetected elements. For example, in Fig. 8a, more than half of the events have an event text. Thus, for the event without an event text (*i.e.*, the event in 2015), we assume it should have an event text. By using

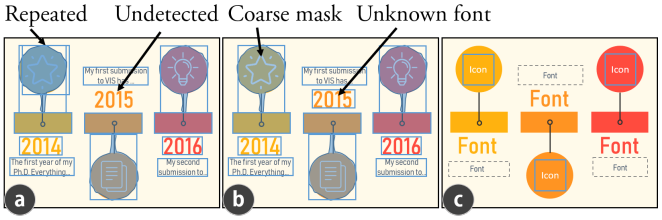

Fig. 8: The reconstruction pipeline: a) use *NMM* and *RR* to eliminate repeated and fix failed detections, respectively; b) use *DL GrabCut* and text recognition to collect the elements to be reused and updated, respectively; c) the final outputs that can be depicted by a specification.

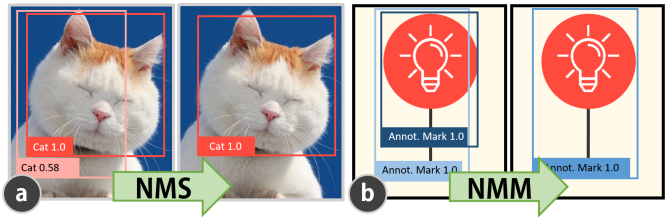

Fig. 9: Eliminate repeated bboxes: a) Non-maximum suppression keeps the bbox with the highest confidence and removes the others; b) Non-maximum merging merges bboxes to the one with the highest confidence and the largest area.

heuristic rules, we can estimate the bbox (*i.e.*, x, y, width, height) of its event text, based on the properties of the event (*e.g.*, parity) and the bboxes of other event texts.

5.3 Elements to be Reused

For an extensible template, certain elements must be reused via segmentation from the infographic image. Our model can predict the pixels (*i.e.*, mask) of each element for segmentation. However, the quality of these predicted pixels (Fig. 10b) may not be accurate enough for template generation.

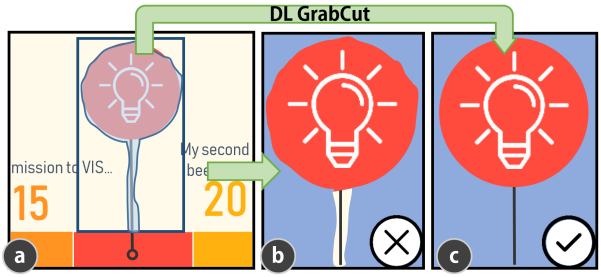

Fig. 10: DL model "interacts" with GrabCut by using the bbox and mask: a) the bbox and mask predicted by our model; b) the predicted mask is coarse; c) the refined result from *DL GrabCut*.

To tackle this issue and obtain high-quality masks of the elements to be reused, we use the outputs from the model (Fig. 10a) as the input to GrabCut [45] algorithm. GrabCut is an interactive segmentation algorithm that has been widely used in production tools, such as Microsoft PowerPoint. It achieves good performance especially when the background and foreground are not similar and the edges of the foreground are smooth, which is a good fit for our scenario. To extract the foreground, GrabCut first needs a bbox around the element to be segmented. Then, it estimates the pixels of the target element by analyzing the color distribution inside and outside the bbox. Thereafter, the user can further refine the segmentation results by drawing stroke to mark probable foreground and background area.

Our idea is to automate this process using the outputs of the model to imitate the user interactions. For each predicted element, we use its bbox as the bbox drew by human and its mask as the user's strokes to refine the segmentation. By this mean, we leverage the semantic information from the DL model and the advantage of GrabCut on image processing to obtain high-quality masks (Fig. 10c).

5.4 Elements to be Updated

Among the six categories of elements, three categories of elements need to be updated, namely, *event text*, *annotation text*, and *annotation icon*. *Annotation icon* can be updated by directly using new icons, whereas *event text* and *annotation text* should maintain the same styles with the original infographic, including their font family, color, and size. To identify the font family, we use Font Identifier powered by Fontspring Matcherator [2]. The font size and color can be calculated and extract from the pixels of the text in the bitmap image. Some annotation text contains title and body text. We heuristically identify the text with the larger font size as the title and the smaller one as the body. To improve the extensibility of the template, in the same spirit as [40], we further use OCR engine (*i.e.*, Tesseract [51]) to recognize the text content of *event text* to infer the visual encodings of the timeline. The final outputs can be depicted using a structural document (Fig. 11).

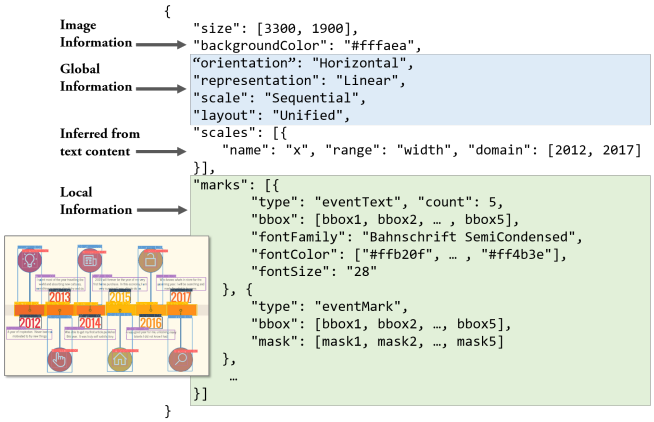

Fig. 11: The extensible template can be organized in a structural document. The **bbox** is a tuple of $(top, left, width, height)$. The **mask** is a byte array with shape $width \times height \times RGB$.

5.5 Validation

To evaluate the effectiveness of our reconstruction pipeline, we reuse the model trained in Sect. 4.4. We are interested in two aspects of our pipeline: whether it can correct the incorrect predictions and whether it can properly segment the elements to be reused from the image. We only test our pipeline on D_2, because the prediction results (both the predicted bboxes and masks) on D_1 are good enough to skip the steps that we want to access.

We use the trained model to parse the local information of the timeline images in the test set of D_2. To control the prediction outputs, we selected the confidence level per category based on the precision-recall curve calculated in Sect. 4.4, ranging from 0.6 to 0.9. We calculate the precision and recall of the predictions with *IoU* at 0.5 and 0.75. We then apply our pipeline on the predictions and calculate the gains of precision and recall on each step. We expected the following results:

- *NMM* can improve the precision of bbox and mask predictions as it removes a few false positives.
- *RR* can improve the precision and recall of bbox and mask prediction as it increases the number of true positives.
- *DL GrabCut* can improve the precision and recall of mask prediction as it improves the quality of masks.

Table 6: Gains come from Reconstruction at *IoU* 0.5 amd 0.75.

	BBox				Mask			
	Pre_{50}	Rec_{50}	Pre_{75}	Rec_{75}	Pre_{50}	Rec_{50}	Pre_{75}	Pre_{50}
Raw	82.9	80.8	74.0	72.1	85.7	81.5	75.8	72.2
+NMM	+2.3	+1.0	+2.3	+0.8	+1.9	+0.6	+1.8	+0.3
+RR	+1.6	+2.5	+1.6	+2.3	+2.3	+2.1	+2.3	+2.0
+DLGC	0.0	0.0	0.0	0.0	-2.8	-5.5	-4.1	-3.8
Total	86.8	84.1	77.9	75.4	84.0	78.4	75.9	71.0

Table 6 presents the results of the gains on precision and recall after each step. As indicated in the table, the gains at *IoU* 0.5 are close to those at *IoU* 0.75. which means the gains from the reconstruction pipeline are strict and stable. As expected, the *NMM* shows a gain on the precision of bboxes and masks predictions. We also observer a small gain on recall. The analysis results indicate that such small gain is attributed to the merging results increasing the number of true positives in some cases (*e.g.*, two false positives become one true positive after merging). Moreover, *RR* shows a gain on the recall of bboxes and masks predictions. These results confirm that our technique can correct some wrong predictions from model outputs.

A surprising finding is the decrease in the precision and recall of mask predictions decrease. Our investigation reveals that this decrease is due to the manually labeled ground truth masks being imperfect.

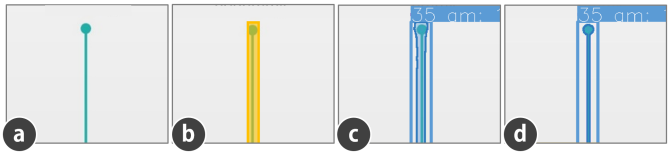

Fig. 12: The error from the imperfect ground truth label. a) an annotation mark and b) its manually labeled ground truth; c) the predicted mask of the annotation mark; d) the refined result from *DL GrabCut*.

Figures 12a and 11b show an annotation mark and its ground truth label. Figure 12c presents the prediction result. while Fig. 12d shows the result returned by *DL GrabCut*. The manually labeled mask encapsulates the graphical mark with empty spaces and a border. Meanwhile, the result returned by *DL GrabCut* perfectly matches the graphical mark but not the ground truth. Thus, even the results from *DL GrabCut* are of high quality and can be used for the extensible template, they can also be changed from a true positive to a false positive, leading to a decrease in precision and recall. Hence, we manually verify the results of *DL GrabCut* and confirm that it can properly segment elements.

6 Extracted Results and Generated Examples

The examples of extracted templates are visualized in Fig. 13, using category labels, bboxes, and masks. Our approach can extract templates from not only the timelines with *linear* representations (*e.g.*, horizontal Fig. 13b, c, e, and vertical Fig. 13f, g), but also those with *arbitrary* representations (Fig. 13a, d, h). Figure 13e shows that our approach is not affected by the background image.

To present the usage of our extensible templates, we further implement a timeline renderer by extending TimelineStoryteller [10] in generating new timeline infographics on the basis of the extracted templates and new data. Figure 14 presents some examples. TimelineStoryteller is an open source tool that allows users to upload their data to generate timelines. It embeds a collection of heuristics to render timelines based on the timeline *type* chosen by users. For example, for a timeline with a *radial* representation, it renders the event marks using polar coordinates; for a timeline with a *faceted* layout, it groups the event data by faceted and renders multiple timelines. Moreover, this tool only provides a set of default styles (*e.g.*, rectangle marks). We reuse and extend the heuristics in the tool and adapt it to our templates. When a user selects a template to render event data, we can automatically render the data and embellish them with the marks in the template because our templates include the types of timelines and the roles of elements. We also add some heuristic rules for effectively using marks in templates, such as looping through the marks when the number of events exceeds that of the marks.

7 Discussion and Future Work

Emerging research has employed deep learning to *understand* visualization, which entails classifying chart types [26], measuring the readability of graph layouts [21], and perceiving visual channels in visualization [20]. However, such research mainly focuses on the use of deep learning in entire images (*i.e.*, understanding global information). To the best of our knowledge, we are the first to apply deep learning to the detection of basic elements in infographics (*i.e.*, understanding local details). We first share the lessons learned from some wrong cases in our study, which outlines the need for *graphical image-driven deep learning*. Then, we discuss how our work can serve as a basis for future research on automated infographic design, and acknowledge the limitations of our study.

7.1 Graphical Image-Driven Deep Learning

The deep learning revolution is driven by tasks on natural images. However, the specificity of graphical images (*e.g.*, charts and infographics) leads to requirements that cannot be easily fulfilled by models designed for natural images. We share the lessons learned from our study and hope to inspire more future work on the fundamental designs of deep learning models.

Translation invariance *vs.* translation variance In some cases, our model cannot distinguish event marks from annotation marks, when they look identical. Although our reconstruction pipeline can fix such incorrect classification in most cases by using Redundancy Recovery, we note that this issue is caused by a key feature of CNNs, namely, *translation invariance*. Translation invariance [19] enables a CNN to recognize an object wherever it is displayed in an image. This feature is important in recognizing natural elements (*e.g.*, a cat should always be classified as "cat" wherever it is displayed). However, it is difficult to handle graphical images, as some graphical elements are translation-invariant while others are translation-variant. For instance, in a bar chart image, the bars in the plot area should always be classified as "bar mark", which requires translation invariance; by contrast, text labels' roles are usually determined by their positions (*e.g.*, "y-axis label" at the left and "x-axis label" at the bottom) and thus requires translation variance. A possible solution to this problem is to learn and recognize relationships among elements. Capsule network [46] is a network structure that can learn the relationships among elements. Further investigation is required to adapt it to graphical images.

High-level semantics *vs.* low-level semantics Our network can predict the pixel-wise masks of elements, but their quality is far from perfect. The problem is rooted in the difference between natural and graphical elements. In general, natural elements do not have smooth edges. Thus, most of the models are designed to use low-resolution,

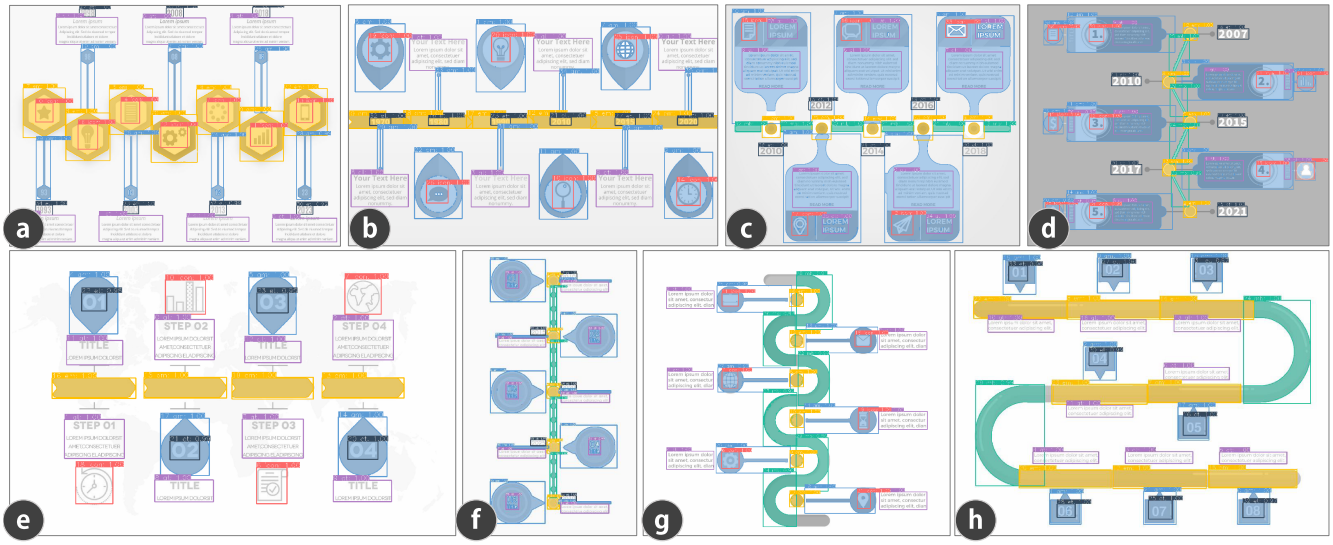

Fig. 13: Examples from our approach. We visualize the final predicted category, bbox, and mask of each element on timelines. We convert the original timeline infographics to gray-scale images for a clear demonstration. The original images and extra results can be checked on our project page.

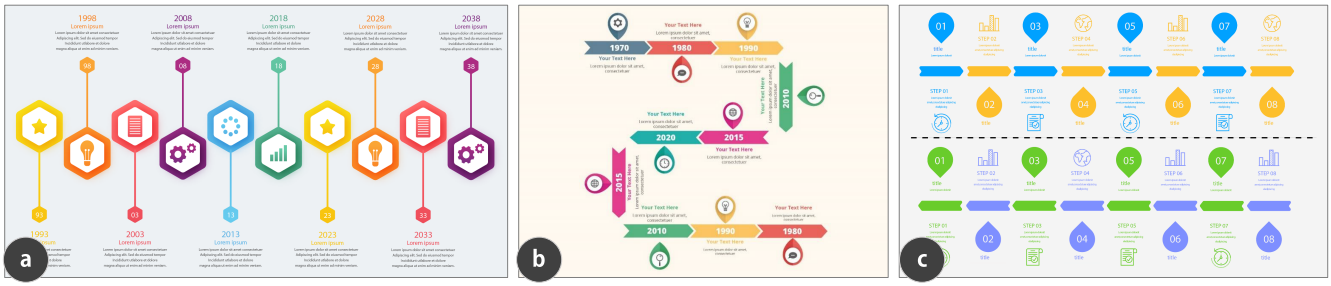

Fig. 14: Automatically generated examples based on the extracted templates and new data. a) A timeline generated using the template from Fig. 13a with updated data. b) A timeline with elements from Fig. 13b and the layout from Fig. 13h. c) A timeline with a multi-faced *layout* generated using the template from Fig. 13e.

semantically-strong features for improved detection, while the precision of segmentation is compromised. By contrast, graphical elements require precise segmentation because of their smooth edges, while high-level semantics are still necessary for detection. This demands a high-resolution, semantically strong features, which is non-trivial to attain. One possible future direction is to use various features for various purposes: low-resolution, semantically strong features for detection; and high-resolution, semantically weak features for segmentation.

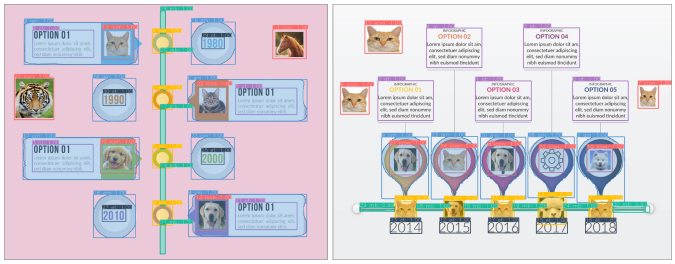

Fig. 15: The animals can be correctly identified as timeline elements.

Single *vs.* hybrid Another difference between natural and graphical images is that graphical images can comprise natural elements and graphical elements. For example, in an infographic, a common practice is to show objects with photos and annotate them with graphical shapes. Such kind of hybrid components requires a model that considers the characteristics of natural and graphical elements. However, some of these characteristics may lead to conflict design requirements, and result in challenges in design models. Although our datasets do not include natural elements, we are interested in the performance of our model on timelines contain graphical and natural elements. Thus, we randomly substitute some graphical marks with photos of animals and then feed them to our model. The results show that our model can still correctly classify the categories of these animals (Fig. 15). We regard this performance as a benefit of the pre-trained network. Future research is needed to further understand the generality of these cases.

7.2 Towards Automated Infographic Design

We propose an end-to-end approach to automatically extract extensible templates from bitmap infographic timelines. This work is only a starting point toward the "holy grail" of automated infographic design, the end-to-end models. Here, we discuss some promising opportunities that can facilitate the design process of infographics.

From timeline to the others. Although our approach focuses on infographic timelines, it can be generalized to other cases. For example, our approach could be adapted to the extraction of visual encodings and data from charts, which are more structural than infographic timelines. For example, we can train our model on a charts dataset to classify chart types, localize elements, and identify element roles to further recover the visual encodings and data. Furthermore, our approach is applicable to other types of infographics, rather than timelines, thereby enabling the indexing of infographics, retargeting of styles, and analysis of content. For example, we can define the categories of graphical marks on other infographics and then use our model to detect them. Thereafter, we can employ *DL GrabCut* to obtain high-quality masks. We plan to extend the application scenarios of our approach in the future.

From hybrid to purely learning-based. Although our approach utilizes a heuristic pipeline, it can be improved by substituting the pipeline with a deep learning model. Our work shows that after extracting the features of an input image, we can decode different image information (*e.g.,* global and local information) by using multi-functional heads. Given that an extensible template can be represented by a structural document (similar to Vega specifications), a potential improvement is to use a recurrent neural network (RNN) to decode the feature maps and directly output extensible templates. Recently, Dibia and Demiralp [15] showed the possibility of translating a JSON-encoded data into Vega-lite specification by using a RNN. Research in computer vision field also presents models that take images as inputs and return textual data as outputs. The related work suggests the potential to extend our model to an end-to-end model, which takes infographic images as inputs and directly outputs templates. We consider this area as an important future direction.

From template-based to freeform. Lastly, our model shows the ability to learn and understand the content of infographic images. This characteristic indicates several potential directions to facilitate the design process. For example, we can use a trained model to interpret a sketch or materials (*e.g.*, data, icons, and textual description) from users and recommend infographic templates. Another step in this direction in this direction is to design mixed initiative authoring systems, including automatically completing or generating designs on the basis of users' sketch or feedback.

7.3 Limitations

We acknowledge the limitations of our study. First, our datasets are rather limited. Our real-world dataset contains 1138 (419 after cleaning) images from three websites, which is relatively small. However, collecting high-quality infographic datasets is not an easy task considering the manual labeling efforts. We plan to open source our datasets and labeling tools for the community and collect larger-scale infographic datasets in the future. Second, given our work is not aimed at high metric values, we did not optimize our model with bells and whistles, including multi-scale train/test, OHEM [49], and other techniques. Outside the scope of this work, we expect that such improvement skills are applicable to our model. Third, although our approach can automatically extract templates from infographic timelines, its performance can be further improved by involving users' refinements. For example, we can integrate our approach to infographic authoring tools and thus allow users to interactively refine extracted results. Copyrights should also be obtained when using our approach on infographics.

8 Conclusion

We contribute an automated approach to extract extensible templates from bitmap infographic timelines. A multi-task DNN is presented to understand and deconstruct bitmap timeline infographics, by classifying the types and orientations of timelines and detecting and segmenting elements on timelines; from these results, a heuristic pipeline is used to reconstruct extensible templates. The extensible templates can be used to automatically generate timeline infographics with updated data. The quantitative experiments and example results confirm the effectiveness and usefulness of our approach. We share lessons learned from our study which make us notice the needs of *graphical image-driven deep learning*. We also discuss how our work can be extended towards automated infographics design in future researches.

REFERENCES

[1] Common objects in context. http://cocodataset.org.

[2] FontsSringfont matcherator. https://www.fontspring.com/matcherator.

[3] FreePik. https://www.freepik.com.

[4] Google image. https://www.google.com/imghp.

[5] Pinterest. https://www.pinterest.com.

[6] Pytorch. https://pytorch.org.

[7] J. Bertin. *Semiology of Graphics*. University of Wisconsin Press, 1983.

[8] A. Bigelow, S. Drucker, D. Fisher, and M. Meyer. Iterating Between Tools to Create and Edit Visualizations. *IEEE TVCG*, 2017.

[9] M. A. Borkin, Z. Bylinskii, N. W. Kim, C. M. Bainbridge, C. S. Yeh, D. Borkin, H. Pfister, and A. Oliva. Beyond Memorability: Visualization Recognition and Recall. *IEEE TVCG*, 2016.

[10] M. Brehmer, B. Lee, B. Bach, N. H. Riche, and T. Munzner. Timelines Revisited: A Design Space and Considerations for Expressive Storytelling. *IEEE TVCG*, 2017.

[11] Z. Bylinskii, S. Alsheikh, S. Madan, A. Recasens, K. Zhong, H. Pfister, F. Durand, and A. Oliva. Understanding Infographics through Textual and Visual Tag Prediction. *arXiv*, 2017.

[12] Z. Bylinskii, N. W. Kim, P. O'Donovan, S. Alsheikh, S. Madan, H. Pfister, F. Durand, B. Russell, and A. Hertzmann. Learning Visual Importance for Graphic Designs and Data Visualizations. In *Proc. UIST*. ACM, 2017.

[13] W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *J. Am. Stat. Assoc.*, 1984.

[14] J. Deng, W. Dong, R. Socher, L. jia Li, K. Li, and L. Fei-fei. Imagenet: A Large-scale Hierarchical Image Database. In *Proc. CVPR*. IEEE, 2009.

[15] V. Dibia and C. Demiralpm. Data2Vis : Automatic Generation of Data Visualizations Using Sequence to Sequence Recurrent Neural Networks. *arXiv*, 2018.

[16] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes Challenge: A Retrospective". *Springer IJCV*, 2015.

[17] R. Girshick. Fast R-CNN. In *Proc. ICCV*. IEEE, 2015.

[18] R. Girshick, J. Donahue, T. Darrell, J. Malik, and U. C. Berkeley. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR*. IEEE, 2014.

[19] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[20] D. Haehn, J. Tompkin, and H. Pfister. Evaluating 'Graphical Perception' with CNNs. *IEEE TVCG*, 2018.

[21] H. Haleem, Y. Wang, A. Puri, S. Wadhwa, and H. Qu. Evaluating the Readability of Force Directed Graph Layouts: A Deep Learning Approach. *CGA*, 2018.

[22] S. Haroz, R. Kosara, and S. L. Franconeri. ISOTYPE Visualization C Working Memory , Performance , and Engagement with Pictographs. In *Proc. CHI*. ACM, 2015.

[23] L. Harrison and K. Reinecke. Infographic Aesthetics : Designing for the First Impression. In *Proc. CHI*. ACM, 2015.

[24] K. He, R. B. Girshick, and P. Dollár. Rethinking imagenet pre-training. *arXiv*, 2018.

[25] K. Z. Hu, M. A. Bakker, S. Li, T. Kraska, and A. Hidalgo. VizML : A Machine Learning Approach to Visualization Recommendation. *arXiv*, 2018.

[26] D. Jung, W. Kim, B. Lee, B. Kim, and J. Seo. ChartSense: Interactive Data Extraction from Chart Images Hyunjoo Song 1 Jeong-in. In *Proc. CHI*. ACM, 2017.

[27] K. Kafle, B. Price, S. Cohen, and C. Kanan. DVQA: Understanding Data Visualizations via Question Answering. In *Proc. ECCV*. Springer, 2018.

[28] Kaiming, He and Georgia, Gkioxari and Piotr, Dollar and Ross, Girshick. Mask R-CNN. In *Proc. ICCV*. IEEE, 2017.

[29] A. Kembhavi, M. Salvato, E. Kolve, M. Seo, H. Hajishirzi, and A. Farhadi. A Diagram is Worth a Dozen Images. In *Proc. ECCV*. Springer, 2016.

[30] N. W. Kim, E. Schweickart, Z. Liu, M. Dontcheva, W. Li, J. Popovic, and H. Pfister. Data-Driven Guides : Supporting Expressive Design for Information Graphics. *IEEE TVCG*, 2017.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Proc. NIPS*, 2012.

[32] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond Sliding Windows: Object Localization by Efficient Subwindow Search. In *Proc. CVPR*. IEEE, 2008.

[33] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proc. CVPR*. IEEE, 2017.

[34] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal Loss for Dense Object Detection. In *Proc. ICCV*, 2017.

[35] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-y. Fu, and A. C. Berg. SSD : Single Shot MultiBox Detector. In *Proc. ECCV*. Springer, 2016.

[36] Y. Luo, X. Qin, N. Tang, and G. Li. DeepEye : Towards Automatic Data Visualization. In *Proc. ICDE*. IEEE, 2018.

[37] J. Mackinlay. Automating the Design of Graphical Presentations of Relational Information. *ACM TOG*, 1987.

[38] J. D. Mackinlay, P. Hanrahan, and C. Stolte. Show Me: Automatic presentation for visual analysis. *IEEE TVCG*, 2007.

[39] D. Moritz, C. Wang, G. L. Nelson, H. Lin, A. M. Smith, B. Howe, and J. Heer. Formalizing Visualization Design Knowledge as Constraints : Actionable and Extensible Models in Draco. *IEEE TVCG*, 2019.

[40] J. Poco and J. Heer. Reverse-Engineering Visualizations: Recovering Visual Encodings from Chart Images. In *Proc. EuroVis*, 2017.

[41] J. Poco, A. Mayhua, and J. Heer. Extracting and Retargeting Color Mappings from Bitmap Images of Visualizations. *IEEE TVCG*, 2018.

[42] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *Proc. CVPR*. IEEE, 2015.

[43] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Network. In *Proc. NIPS*, 2015.

[44] S. F. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein. Interactive Graphic Design Using Automatic Presentation Knowledge. In *Proc. CHI*. ACM, 1994.

[45] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut": Interactive Foreground Extraction Using Iterated Graph Cuts. *ACM TOG*, 2004.

[46] S. Sabour, N. Frosst, and G. Hinton. Dynamic Routing between Capsules. In *Proc. NIPS*, 2017.

[47] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite : A Grammar of Interactive Graphics. *IEEE TVCG*, 2018.

[48] 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. UIST*. ACM, 2011.

[49] A. Shrivastava, A. Gupta, and R. B. Girshick. Training Region-based Object Detectors with Online Hard Example Mining. In *Proc. CVPR*. IEEE, 2016.

[50] N. Siegel, Z. Horvitz, R. Levin, S. Divvala, and A. Farhadi. FigureSeer: Parsing Result-Figures in Research Papers 1 Computer Vision for Scholarly Big Data. In *Proc. ECCV*. Springer, 2016.

[51] R. Smith. An overview of the tesseract ocr engine. In *ICDAR*, 2007.

[52] C. STEPHEN M. A Task-Analytic Approach to the Automated Design of Graphic Presentations. *ACM TOG*, 1991.

[53] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. In *Proc. CVPR*. IEEE, 2015.

[54] Y. Wang, H. Zhang, H. Huang, X. Chen, Q. Yin, Z. Hou, D. Zhang, Q. Luo, and H. Qu. InfoNice: Easy Creation of Information Graphics. In *Proc. CHI*. ACM, 2018.

[55] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager : Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE TVCG*, 2016.

[56] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2 : Augmenting Visual Analysis with Partial View Specifications. In *Proc. CHI*. ACM, 2017.

[57] H. Xia, N. Henry Riche, F. Chevalier, B. De Araujo, and D. Wigdor. DataInk: Direct and Creative Data-Oriented Drawing. In *Proc. CHI*. ACM, 2018.

[58] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proc. CVPR*. IEEE, 2017.