Gallery D.C.: Design Search and Knowledge Discovery through Auto-created GUI Component Gallery

CHUNYANG CHEN, Monash University, Australia
SIDONG FENG and ZHENCHANG XING, Australian National University, Australia
LINDA LIU, Google, China
SHENGDONG ZHAO, NUS-HCI Lab, Singapore
JINSHUI WANG, Fujian University of Technology, China

Online communities like Dribbble and GraphicBurger allow GUI designers to share their design artwork and learn from each other. These design sharing platforms are important sources for design inspiration, but our survey with GUI designers suggests additional information needs unmet by existing design sharing platforms. First, designers need to see the practical use of certain GUI designs in real applications, rather than just artworks. Second, designers want to see not only the overall designs but also the detailed design of the GUI components. Third, designers need advanced GUI design search abilities (e.g., multi-facets search) and knowledge discovery support (e.g., demographic investigation, cross-company design comparison). This paper presents Gallery D.C. (http://mui-collection.herokuapp.com/), a gallery of GUI design components that harness GUI designs crawled from millions of real-world applications using reverse-engineering and computer vision techniques. Through a process of invisible crowdsourcing, Gallery D.C. supports novel ways for designers to collect, analyze, search, summarize and compare GUI designs on a massive scale. We quantitatively evaluate the quality of Gallery D.C. and demonstrate that Gallery D.C. offers additional support for design sharing and knowledge discovery beyond existing platforms.

CCS Concepts: • **Information systems → Data mining**; • **Human-centered computing → Systems and tools for interaction design**;

Additional Key Words and Phrases: Mobile Application; GUI Design; Object Detection; Multi-faceted Design Search; Design Demographics; Design Comparison

ACM Reference Format:
Chunyang Chen, Sidong Feng, Zhenchang Xing, Linda Liu, Shengdong Zhao, and Jinshui Wang. 2019. Gallery D.C.: Design Search and Knowledge Discovery through Auto-created GUI Component Gallery. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 180 (November 2019), 22 pages. https://doi.org/10.1145/3359282

1 INTRODUCTION

Graphical User Interface (GUI) design is a creative knowledge work. Design sharing is a routine activity through which design knowledge, creativity and aesthetics is exchanged among designers. GUI designs can be shared in the form of reusable design templates or design kits [1–3, 5, 7] (see

Authors' addresses: Chunyang Chen, Faculty of Information Technology, Monash University, Melbourne, Australia, chunyang.chen@monash.edu; Sidong Feng; Zhenchang Xing, Research School of Computer Science, Australian National University, Canberra, Australia, u6063820@anu.edu.au,zhenchang.xing@anu.edu.au; Linda Liu, Google, Beijing, China, lindahliu@google.com; Shengdong Zhao, NUS-HCI Lab, Singapore, Singapore, zhaosd@comp.nus.edu.sg; Jinshui Wang, corresponding author, Fujian University of Technology, Fuzhou, China, ymkscom@gmail.com.

2573-0142/2019/11-ART180 $15.00
https://doi.org/10.1145/3359282

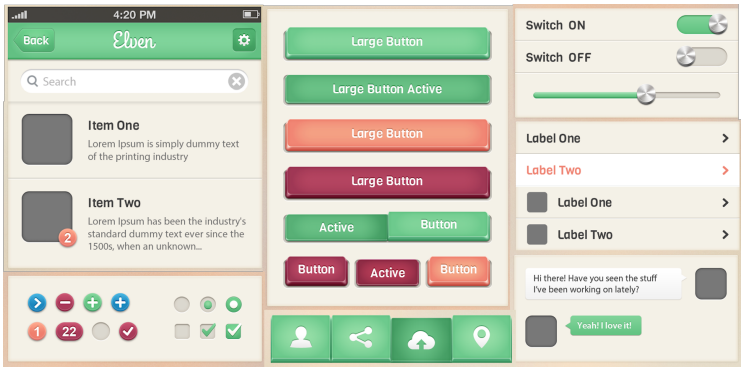
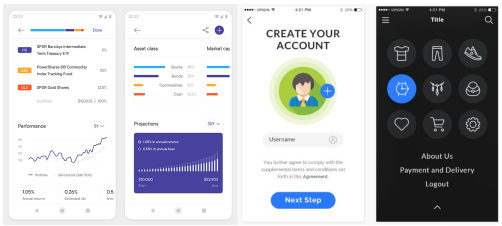
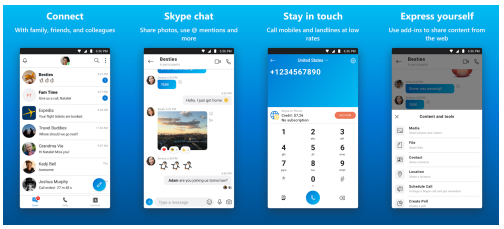


Fig. 1. Examples of design kits of GUI components [1]



(a) Examples of GUI design artwork (left two shared on Dribbble and right two shared on GraphicBurger



(b) Examples of application introduction screenshots of Skype on Google Play

Fig. 2. Differences between app runtime GUI screenshots and app introduction GUI screenshots

Fig. 1 for examples). With the advent of Web 2.0, online platforms allow GUI designers to share their design artwork in online communities. For example, Fig. 2(a) shows some GUI designs shared on Dribbble and GraphicBurger. Through design sharing, the designers can gain feedback from the community and boost their portfolio. Meanwhile, the repository of the shared GUI designs provides an opportunity to learn about GUI designs, gain design inspiration and understand design trend.

Despite of all these benefits, existing design sharing platforms have three fundamental limitations in terms of *design practicality*, *design granularity* and *design knowledge discovery*. First, existing design sharing platforms showcase creative design artworks, but GUI designers also need to see the practical use of certain design ideas in real applications. As illustrated in Fig. 7, GUI design artwork and real application GUIs may have substantial differences. Second, existing design sharing platforms tend to support the sharing of the overall design of the GUI only. Although there are some GUI component design kits (e.g., examples in Fig. 1), these component design kits do not provide the GUI context in which certain components can be practically composed. In the design work, designers need to drill down specific GUI components and at the same time still need to see the composition of GUI components as a whole. Third, existing design sharing platforms support only tag-based search of the GUI designs. However, designers also need advanced design knowledge discovery, for example, multi-faceted design search (e.g., by component types, application categories, color, size, displayed text), summarization of design demographics (e.g., color and size distribution), and design comparison across competitors. We will further elaborate these knowledge discovery needs with examples in Section 7.

With millions of real-world applications - each comprising multiple UI designs of human creativity and aesthetics - the application market provides an opportunity to learn about practical GUI designs on a truly massive scale, even though the GUI of these applications are not created for the purpose of design sharing and knowledge discovery. However, collecting and inspecting real-application GUIs in an ad-hoc fashion cannot effectively exploit this gold mine of GUI design resources. This paper presents an automatic approach to turn GUI designs implicitly shared by millions of application developers into a large-scale GUI component gallery, on top of which advanced design search and knowledge discovery can be performed. We refer to our techniques to build this design gallery as invisible crowdsourcing of GUI design resources, as opposed to intentional design sharing on Web 2.0 platforms.

We build this GUI component design gallery using reverse-engineering and computer vision techniques. In particular, we design an automated GUI exploration method to automatically collect at runtime 68,702 GUI screenshots of 5,043 Android applications and their metadata (i.e., component type, size and position). One way to collect such data is through automated GUI exploration by setting up Android application emulator and execute the applications to collect GUI screenshots at runtime; however, this process is time consuming and not very scalable.

To obtain more GUI components from high-quality GUI designs, we use the GUI design dataset obtained through automated GUI exploration to train a deep-learning object detection model (Faster RCNN [30]) that can automatically wirify (i.e., decompose) a GUI design image into a set of GUI components and determine the type, size and position of the GUI components in the GUI design image. We then crawl 469,177 application introduction screenshots of 126,298 Android applications in Google Play [4]. The application introduction screenshots (see examples in Fig. 2(b)) usually illustrate the most important features and the best-designed GUIs of an application. To enhance the model training, we develop a GUI-specific image-augmentation method to transform the application runtime screenshots into similar style of application introduction screenshots. We use the trained object-detection model to obtain the GUI components in the application introduction screenshots. In addition to the component metadata, we also detect the primary color of GUI components in HSV color space [27], and collect app metadata such as app category and download times.

Based on the collected application runtime and introduction GUI screenshots, the GUI components in these screenshots, and the application and GUI component metadata, we build a web application *Gallery D.C.* (http://mui-collection.herokuapp.com/) for GUI designers and developers to access the large-scale GUI and GUI component designs. The users can perform multi-faceted search by component types, size, color, application category and/or displayed text. The system can report the design demographics of the GUI designs in the entire gallery or in the design search results. The system also allows users to select and visually compare the GUI designs of different companies in a comparative shopping manner. These design search, summarization and comparison features enable advanced design knowledge discovery of real-world application GUI designs on a truly massive scale, beyond the content sharing of GUI design artworks. Our contributions can be summarized below:

- We identify the fundamental limitations of existing design sharing platforms and propose to exploit real-world application GUIs implicitly shared in the application market to address these limitations.
- We develop reverse-engineering and computer-vision based techniques to automatically transform half a million GUI screenshots of over 130,000 Android applications into a large-scale GUI component design gallery, which contains design creativity and knowledge of millions of application designers and developers.

- Our GUI component design gallery enables invisible crowdsourcing of GUI design resources and new ways of design sharing and design knowledge discovery beyond content sharing through mining massive-scale GUIs of real-world applications.

2 APPROACH OVERVIEW

To address the information needs of designers beyond simple content sharing and search, we design innovative ways to collect, analyze, search, summarize and compare GUI designs of real-world applications on a truly massive scale. In this section, we introduce our problem definition and give an overview of main steps of our approach.

2.1 Problem Definition: A Large-Scale Real-World Application GUI Components Design Gallery

A mobile platform, such as Android or iOS, supports a wide range of GUI components for building the GUI of mobile apps. In this work, we explain and evaluate our approach using Android apps, but our approach is not limited to just Android apps. Our goal is to build a large-scale gallery of 11 types of GUI components for Android GUI design. These 11 types of UI components can be divided into two general categories for easy reference: *button* (including button, image button, check box, radio button, switch, and toggle button), and *information bar* (progress bar, rating bar, seek bar, spinner, and chronometer). Fig. 3 shows some examples of these 11 types of GUI components that are wirified from Android app introduction GUI screenshots using our approach. As seen in these examples, designs of a type of GUI components can vary greatly in component size, color and other visual effects. Being able to easily access the design of a type of GUI components from thousands of Android apps would help designers and developers understand the landscape and practicality of GUI designs for their own design tasks. Note that we do not consider content-centric UI components such as text view and image view, because these component is highly content dependent without much consideration of component design itself.

2.2 Main Steps: App GUI Screenshot Collection, GUI Component Wirification, Gallery System Design and Construction

To build a large-scale GUI component design gallery from existing Android apps, our approach involves three steps: app GUI screenshot collection (Section 3), GUI component wirification (Section 4), and gallery system design and construction (Section 5).

App GUI screenshot collection. This step addresses the designers' information need for design practicality. We collect two kinds of app GUI screenshots for the whole work. The first dataset contains 68,702 GUI screenshots of 5,043 Android apps. We download the Android apps from Google Play. Based on the Android app GUI testing framework [31], we develop an automated UI exploration method to automatically execute an app, explore the app's GUIs, take the runtime GUI screenshot and also dump the UI components and their metadata (component type, size and position) in the screenshot. That is, the collected screenshots are automatically annotated with the information of GUI components they have. Note that although automated GUI exploration can collect many GUI components, it requires setting up proper Android emulator to execute the apps and takes a long time to sufficiently explore an app's GUIs. It is also difficult to ensure the design quality of the collected GUI components, as the GUI exploration is random. Therefore, the dataset collected through automated GUI exploration is mainly for training GUI component wirification model. The second dataset contains 469,177 app introduction GUI screenshots of 126,298 Android apps crawled from Google Play. Crawling app introduction screenshots from Google Play can be easily done for a very large number of apps in Google Play. Furthermore, app introduction screenshots illustrate the most important GUIs (usually the best-designed GUIs) of an app. So the

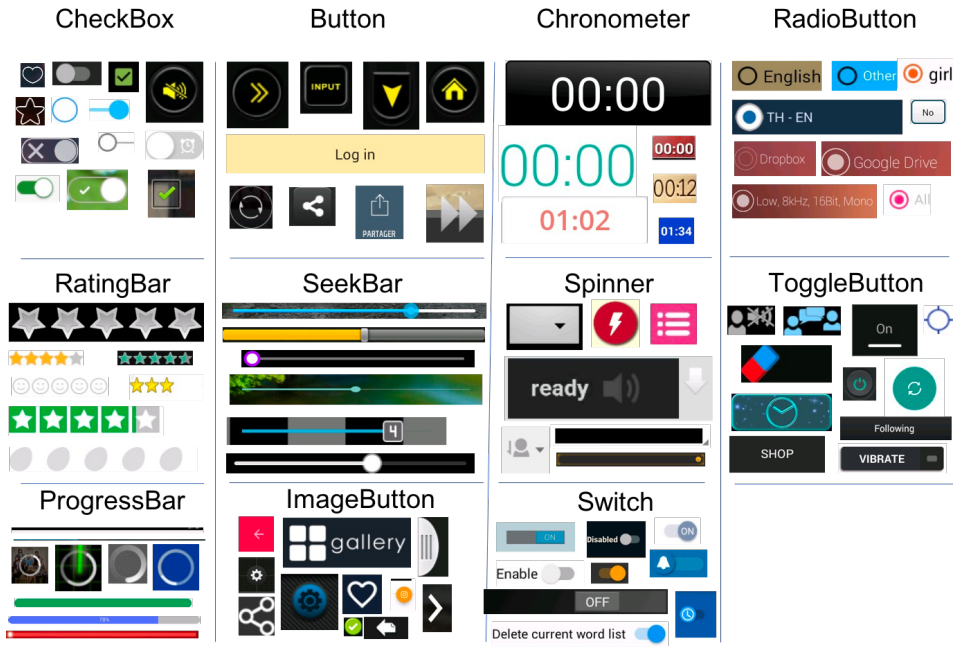


Fig. 3. Examples of 11 types of UI components wirified in app introduction screenshots by our approach.

second dataset is the main source of the high-quality GUI components in our component design gallery.

GUI component wirification. This step addresses the designers' information need for linking the two design granularities: the whole GUI design and the GUI component design. We use the first dataset (i.e., the annotated runtime GUI screenshots) to train the GUI component detection model using Faster RCNN [30]. The trained model is then used to wirify (i.e., decompose) the app introduction screenshots in the second dataset into the GUI components (with component metadata) in these screenshots. We design a GUI-screenshot specific image augmentation method to make runtime GUI screenshots in the first dataset similar to the app introduction screenshot in the second dataset so that the GUI component detection model trained on the first dataset can be applied to the second dataset.

Gallery system design and construction. This step addresses the designers' information need for advanced design search and knowledge discovery. We compile a large-scale gallery of GUI component designs (together with the original whole GUIs) obtained in the above two steps. To improve the diversity of design gallery, we remove many duplicated or very similar GUI components by comparing their structural similarity (SSIM) index [33]. To facilitate the search of GUI components in the design gallery, we augment component metadata with the primary color of GUI components by HSV colorspace and collect the displayed text (if any) of the GUI components by Optical Character Recognition (OCR) technique. We implement a web-based search interface (http://mui-collection.herokuapp.com/) that supports multi-faceted GUI component search by multi-dimensional component metadata (i.e., app category, app download times, component type (widget class), component height/width and primary color, and displayed text in the component). In addition, the system automatically summarizes several design demographics of the GUI components in the search results, including distributions of component type usage, component primary colors, and component height/width. The gallery system allows designer to select different companies to

compare their GUI component designs. It generates a comparison table (like the one in Fig. 14) for designers to visually compare GUI components side by side.

3 APP SCREENSHOT COLLECTION

First, we describe how we collect app runtime GUI screenshots for training GUI component wirification model, and crawl app introduction GUI screenshots from Google Play for building the GUI component design gallery. Using real-world application GUIs on a truly massive scale exposes designers to not only design creativity and aesthetics but also design practicality.

3.1 Collecting Annotated Runtime App GUI Screenshots by Automated GUI Exploration

The core technique of our GUI component wirification model is a deep learning based object-detection model Faster RCNN [30]. To effectively train the Faster RCNN model, we need to prepare sufficient training data. An instance of the training data is an app GUI screenshot annotated with the GUI component information (including component type, size and position) in the screenshot. To acquire sufficient annotated app GUI screenshots for model training, we develop a data collection tool that automatically executes a mobile app, explore the GUIs of the app by simulating the user's interaction with the app's GUIs, and collect the runtime GUI screenshots and the corresponding GUI component information during the simulated interaction process.

Fig. 4. An illustrative example of simulating user actions on mobile app GUI proceeding from state UI-1 to state UI-4.

The automated UI exploration is inspired by automated UI testing techniques [31]. Users interact with mobile apps by various actions (e.g., click, edit, scroll). Our data collection tool tries to identify executable GUI components (e.g., clickable, long-clickable editable, scrollable) on the current UI and infer actions from the type of these components. It then emits the corresponding UI events to simulate user actions, and automatically explore different parts of a mobile app. The tool can also simulate the interaction with the hardware buttons (e.g., back).

As illustrated in Fig. 4, starting with the UI-1, the tool "scrolls down" to the bottom of the page (UI-2). The tool then "clicks Settings" and the app moves back to the home page (UI-3). Next, the tool "inputs text" for the two EditText boxes transiting the app GUI into the UI-4. Of course, the tool can simulate other user actions on the app GUIs which will explore other parts of the mobile app that are not shown in this illustrative example.

During such automated GUI exploration, the data collection tool uses Android UI Automator [8] to automatically dump pairs of runtime GUI screenshots and corresponding runtime GUI component hierarchies. The runtime GUI component hierarchies are stored in XML files. Fig 5 shows an example

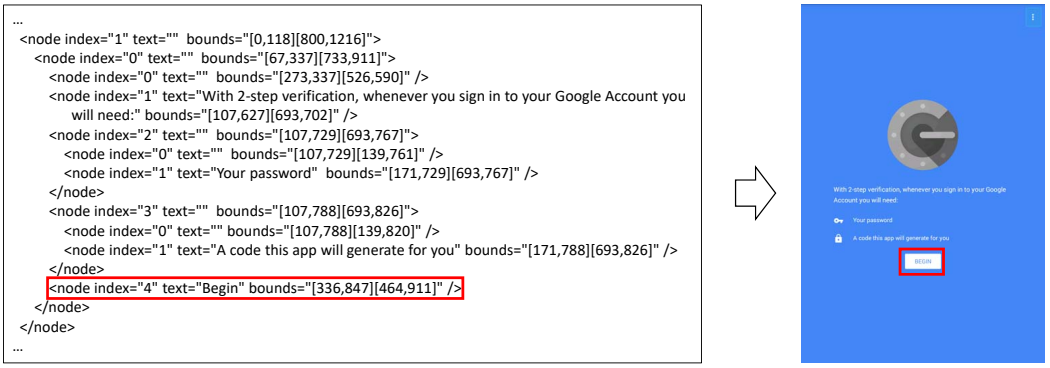


Fig. 5. An illustrative example of the run-time GUI hierarchy and corresponding GUI screenshot

about the UI and corresponding code, and the button outlined in the red box is corresponding to the code also outlined in the red box. We can obtain a list of GUI components in an GUI screenshot from the corresponding GUI component hierarchy. For each UI component, we extract its component type from "class" attribute and its size/position from "bounds" attribute. Optionally, if the "text" attribute is not null, we also extract the value of "text" attribute as the displayed text on the GUI component. Through this automated GUI exploration and data collection process, we are able to collect a large set of app GUI screenshots annotated with the GUI component information.

As shown in Fig. 4, we can model the GUI transitions triggered by the simulated user actions as a state machine. A single activity class as coded by the developer may contain a few different states depending on the strictness of state definition. For example, the UI-1 and UI-2 or the UI-3 and UI-4 in Fig. 4 are implemented in the same activity class but have different GUI components. Furthermore, different UI transitions may lead to the same UI. For example, in addition to UI-1 $\rightarrow$ UI-2 $\rightarrow$ UI-3 transition, "click Settings" on UI-1 can directly go to UI-3. Therefore, we need to discern different UI states. In this work, we adopt the method proposed in [10], which discerns UI states by encoding the underlying component hierarchy of a GUI screenshot.

First, we assign an unique *type_index* to each type of GUI component, for example, Button as 0, TextView as 1, RatingBar as 2, etc. Then, we obtain the *node_index* (starting at 0) of each GUI component in the screenshot from the corresponding GUI component hierarchy and concatenate a component's *node_index* with its *type_index*. For example, the first UI component is a button and it can be represented as "0_0", and the second component is a rating bar and it can be represented as "1_2". In this way, we obtain a representation string for a UI screenshot by concatenating the representation string of all GUI components in the screenshot. We hash the representation string of the GUI screenshot using MD5 algorithm to get a fixed-length unique identifier for the UI screenshot. If the two GUI screenshots have the same identifier, then they will be regarded as the same UI state. During the automated GUI exploration, the data collection tool maintains a list of already-seen UI states and will expand the list if the current GUI screenshot has an unseen identifier.

We randomly download 6000 Android apps from Google Play App Store. We execute the downloaded Android apps on Android emulators (configured with the popular KitKat version, SDK 4.4.2, API level 19, 768-1280 screen size). We run our data collection tool on a 64-bit Ubuntu 16.04 server with 32 Intel Xeon CPUs and 189G memory, and it controls 16 Android emulators (one for each Android app) in parallel to collect annotated GUI screenshot. Each Android app is run for 45 minutes. 5043 crawled apps runs successfully on our data collection platform. These apps belong to

25 categories. The other 957 apps require extra hardware support or third-party libraries which are not available in the Android emulator we use. Our data collection tool collects in total 68,702 app GUI screenshots annotated with the corresponding UI component information (on average about 13.6 UI screenshots per app). This UI dataset is used for training and testing our GUI component wirification model (see Section 4.1).

3.2 Crawling App Introduction Screenshots in Google Play

In Google Play, developers of apps are encouraged to upload GUI screenshots[1] that showcase their app's features and functionality. These screenshots can help apps attract new users on Google Play, and we call them the "app introduction GUI screenshots" to distinguish them from app runtime screenshots collected in the last section. We crawl 469,177 app introduction screenshots of 126,298 Android apps. Note that although both app runtime GUI dataset and app introduction GUI dataset come from Google Play, the apps in the second dataset has few overlap with those in the first dataset. As Google Play only allows uploading eight introduction screenshots at most, the maximum number of GUI screenshots per app in the second dataset is eight. Furthermore, the app introduction screenshots are simply GUI images without GUI component information. We will use the GUI component wirification model to wirify the app introduction GUI screenshots into the GUI components.

4 GUI COMPONENT WIRIFICATION

To obtain the GUI elements from the app introduction screenshots crawled from Google Play, we need a method to decompose (i.e., wirify) app GUI screenshots into GUI components in the screenshots. Through this GUI component wirification step, we naturally link the whole GUI design and the GUI component design.

Our design wirification task is a well-known problem, called "object detection" in Computer Vision domain. In this work, we adopt the Faster RCNN model [30], a mature object-detection technique. To make the Faster RCNN model work well on our GUI screenshot data, we train the model with our dataset of annotated app runtime GUI screenshots (Section 4.1). Different from our goal, existing object-detection models focus on the detection of objects in natural scene images, such as finding cats or humans in the photos. Instead, our goal is to extract GUI components from computer-generated GUI images, including app runtime GUI screenshots, and app introduction screenshots that contain not only GUIs but also other information like natural language descriptions and markers of app features like the examples in Fig. 7. To improve the model performance, we also carry out the image augmentation to annotated app runtime screenshots to mimic the presentation style of app introduction screenshots crawled from Google Play (Section 4.2).

4.1 UI Component Wirification by Deep-Learning based Object Detection

Compared with image classification, the task of object detection is not only to classify objects in the image, but also get the detailed location of the object. The object detection can be modeled as a classification problem where we take windows of different sizes from input image at all the possible locations, and feed these windows to an image classifier to determine objects in them. Recently, the performance of object detection is further boosted by exploiting the deep learning models [19, 20, 30]. In this work, we adopt the state-of-the-art Faster-RCNN [30]. Fig. 6 presents the overall architecture of the model. The model uses a Convolutional Neural Network [22, 24] to extract image features from the input GUI screenshot. It then use a region proposal network (RPN) to generate region proposals that likely contain some object of interest (as opposed to only

[1] https://support.google.com/googleplay/android-developer/answer/1078870

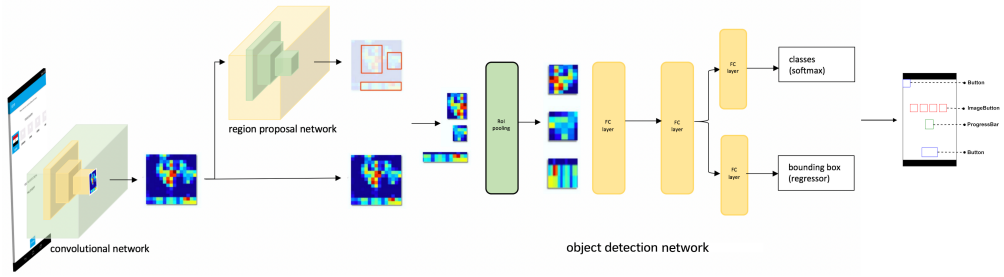


Fig. 6. The architecture of Faster RCNN [30]

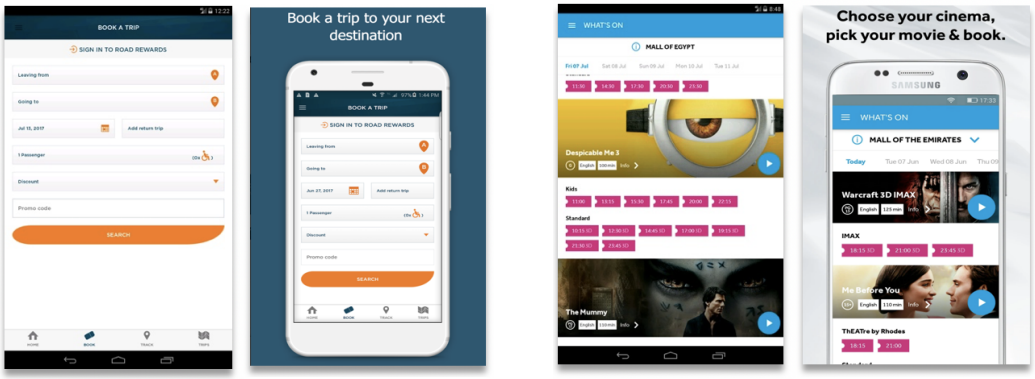


Fig. 7. Differences between app runtime GUI screenshots and app introduction GUI screenshots

background). These regional proposal are called Region of Interest (ROI). Finally, it uses an object detection network that predict object classification scores for the region proposals and determines the object bounding box.

4.2 Enhancing Model Performance by App Screenshots Augmentation

Our GUI component wirification model is trained by the annotated app runtime screenshots collected through automated GUI exploration (see Section 3.1), and we want to apply our model to app introduction GUI screenshots crawled from Google Play. But the app introduction screenshots in Google Play are different from the app runtime screenshot. The app introduction screenshots are more like posters which contain not only the app GUI screenshots, but also much other information. Fig. 7 shows the screenshots of the two apps from the two data sources respectively. For each app, the left image is the runtime GUI screenshot collected by our automatic GUI exploration tool, and the right image is the app introduction image in Google Play. Such image differences between the training data and the testing data may lead to poor performance of the trained model. To bridge the gap between the GUI screenshots from the two resources, we apply the image augmentation method to transform the training screenshots to mimic those in the testing data.

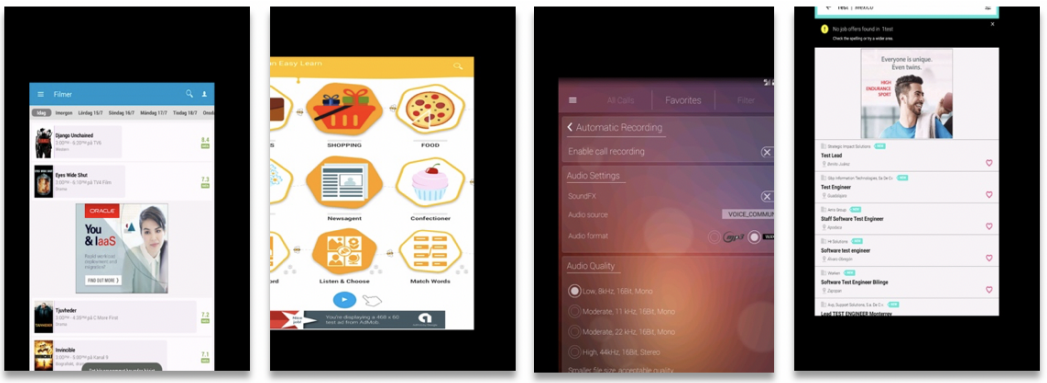


Fig. 8. Examples of resulting GUI screenshots by image augmentation

For machine learning tasks such as image classification and machine translation, the amount of data available for training is critically important to achieve good performance with better generalization. To increase the data, data augmentation [9, 34, 35] is widely used. It mainly applied a small mutation in the original training data to create more new data. For example, the image augmentation methods include cropping, padding, flipping, resizing, inverting colors, blurring, etc. In this work, our target is to mutate app runtime screenshots to make them similar to the apps' introduction images. According to our observation, the app GUI screenshots are always part of the introduction images in Google Play with some words or other highlights around. Therefore, we randomly resized the app runtime GUI screenshots, and move it into four directions randomly (e.g., left, right, up, down) around the center of the image. Some resulting app screenshots by data augmentation can be seen in the Fig 8. Correspondingly, the ground-truth annotation of GUI components will undergo the same transformation to align with the transformed runtime screenshots.

4.3 Post-processing GUI Components Obtained Through GUI Component Wirification

After collecting all detected GUI components from the crawled app introduction screenshots, we further post-processing these GUI components by removing duplicates and also determining their primary colors.

4.3.1 Removing Duplicate UI Components. To keep consistent, one app always use the same GUI components across its GUIs such as the "OK" button, navigation bar, or icons in the title bar. Even different apps may also share very similar GUI components such as Facebook login button, back buttons, etc. Showing similar GUI components repeatedly to designers may be a waste of their time. These similar components will also reduce the diversity and serendipity of the design gallery, and bias the design demographics. So we remove duplicated or very similar elements to purify our design gallery.

To determine how similar two GUI components re, we adopt the structural similarity (SSIM) index. According to the similarity score, we decide if two GUI components are very similar or not. However, removing duplicated elements may cause losing useful elements. Therefore, we set 0.95 as a threshold empirically. The threshold value is carefully selected through manually curating 100 pairs of duplicate screenshots from randomly chosen applications and computing the minimum similarity value. Considering the duplication removal efficiency, we first carry out within-app duplicate removal and then cross-app removal.

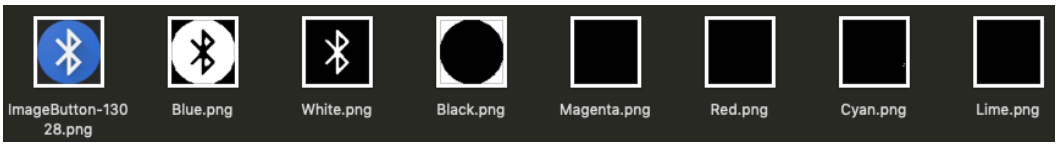


Fig. 9. An example of determining the primary color of GUI component by HSV mask

4.3.2 Determining Primary Color of GUI Components. A usage feature that designers suggest in our interview is to search GUI components by their primary color, because color design is an important aspect of GUI design. In order to achieve that, we need to enhance the initial metadata of GUI components by adding an attribute to keep track of the primary color of the GUI components. To that end, we adopt HSV colorspace for color detection. We first makes a conversion from RGB color to HSV colorspace. Each RGB color has a range of HSV value. The lower range is the minimum shade of the color that will be detected, and the upper range is the maximum shade. For example, black is in the range of $\langle 0, 0, 0\rangle - \langle 184, 254, 80\rangle$. Then, we create a mask for each color (black, blue, cyan, green, lime, megenta, red, white) as shown in Fig. 9. The mask is the areas that HSV value on pixels match the color between the lower range and upper range. Finally, we calculate the area of the mask in each color and the corresponding image occupancy ratio. The color with the maximum ratio is identified as the primary color of the GUI component (the blue in the example in Fig. 9).

5 GALLERY SYSTEM DESIGN AND CONSTRUCTION

With the real-world application GUIs collected in Section 3 and the GUI components (together with the original GUI screenshots) obtained in Section 4, we are now ready to build a novel GUI component design gallery system that can satisfy the designers' information need for advanced design search and knowledge discovery, including multi-faceted design search, design demographics, and design comparison. We implement a proof-of-concept prototype of our GUI component design gallery in the web application http://mui-collection.herokuapp.com/. Note that the approach in our work can be regarded as a kind of reverse engineering process which is allowed by both the US Law[2] and Europe Law[3] which explicitly offer users the right to decompile in order to achieve interoperability.

5.1 Multi-Faceted Search

GUI components fall into a multi-dimensional information space. Multi-faceted search has been shown to be an effective mechanism to search, explore and filter multi-dimensional information space.The successful examples include online shopping and travel booking websites. Inspired by these successes, we also design a multi-faceted search interface for searching GUI component designs. Based on the available application and component metadata, our gallery system supports five search facets: widget class (component type), component primary color, application category, displayed text, and component height/width. In our prototype, we support 11 types of GUI components (see Section 2.1) The component primary color is determined as described in Section 4.3.2. Our dataset currently have 25 categories of Android applications, such as *music_and_audio* category or those for *books_and_reference*. The users can enter a short text, such as "Sign Up", "check out", and the system uses regular expression to match GUI components containing the searched text. The users can search the GUI components within the specified height and/or width range.

[2] https://www.law.cornell.edu/uscode/text/17/1201
[3] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=LEGISSUM%3Ami0016

Users can freely combine the five facets in their search. Some usage scenarios are demonstrated in Section 7.1. Based on the user-specified search facets, the gallery system returns a list of GUI components that match the search criteria. The search results are sorted by descending number of application downloads (by default) or by the alphabetic order of application names. To be responsive, the system uses the lazy loading strategy commonly used in image search engine. It shows some initial search results on the visible screen and loads more search results when users scroll down the search results. The users can select a GUI component in the search results and view the selected component in the original whole GUI screenshot. The user can also see additional facts about the GUI component, the screenshot and the application (e.g., application vendor, download numbers, rating, and release date), as well as similar GUI components for a selected component.

5.2 Design Demographics

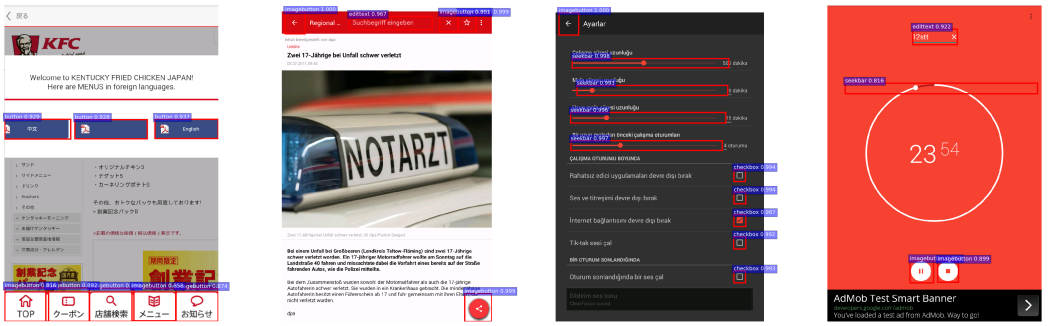
In addition to view individual GUI designs, designers also need to understand the overall design landscape [14, 15] at an aggregation level. Aggregation can be done for the whole design gallery or for a specific design facet or the combination of some facets defined above. In our current prototype, we compute design demographics for the multi-faceted search results so that the users can obtain a quick overview of the key characteristics of the search results. The system dynamically computes four types of design demographics for the search results: the component primary color, the component height and width, the number of components of each component type, the number of component of each application category. The system visualizes the distribution of component primary colors in piechart, the component height and width in scatter plot, the distribution of component numbers by component types in piechart, and the number of components of application categories in bar chart. Some usage scenarios are demonstrated in Section 7.2.

5.3 Design Comparison

Design comparison allows designers to see the brand commonalities and variations of different companies. We borrow the idea of comparison shopping to support design comparison. As our GUI data comes from Google Play, we have the vendor information for each collected GUI. A vendor may have multiple applications whose GUIs have been collected in our gallery. Each application has metadata such as download times and rating. We organize the GUIs by the same vector and rank the vendors by the sum of their applications' download times. Currently, for each vendor, we select four distinct GUI screenshots from all GUI screenshots of this vendor to form a vendor GUI gallery of its representative GUIs. These representative GUIs are determined by image differencing and image clustering. Users can browse this vendor GUI gallery. Furthermore, users can add multiple vendors to a "shopping" cart and request design comparison of these vendors. For each vendor under comparison, the system will select up to six distinct GUI components for each component type (again based on image differencing and image clustering) from all GUI components of this vendor in our gallery. If the vendor does not have a particular type of components, the system reports "None" for this component type. The system will also compute the component primary color and the component height/width demographics for each component type. Finally, the system generates a design comparison table that allows users to visually compare the GUI components and their demographics of the selected vendors side by side. Some usage scenarios are demonstrated in Section 7.3.

6 EVALUATION OF GUI COMPONENT WIRIFICATION MODEL

The key enabling technique for building our design gallery is the deep learning based GUI component wirification model that extracts GUI components from app introduction GUI screenshots (see Section 4). To evaluate the effectiveness of our GUI component wirification model, we test the



(a) Components with texts in different languages (accurately detected)

(b) Components sharing the same background (accurately detected)

(c) Many components (accurately detected)

(d) Oddly shaped seekbar (bounding box is inaccurate)

Fig. 10. Results of GUI component wirification in complex GUI screenshots

trained model not only with the annotated runtime GUI screenshots collected through automated GUI exploration, but also manually check the model's performance on the app introduction GUI screenshots crawled from Google Play.

6.1 Experimental Setups

Among all 68,702 annotated runtime GUI screenshots collected through automated GUI exploration, we perform a 90/10 train/test split (i.e., 61,832 for training the UI component wirification model and 6,870 for testing the trained model). We train our GUI component wirification model with the training data on a Nvidia P40 GPU (24G memory) with 140k iterations for about two days. Apart from testing the model with annotated runtime GUI screenshots, we also test our model on 100 randomly sampled app introduction screenshots crawled from Google Play.

6.2 Evaluation Metrics

For each region proposal r, our model outputs two vectors; softmax probability p and per-class bounding box regression offsets ranked by the corresponding confidence output. A detection confidence is assigned to r for each GUI component class k using the estimated probability $P(class = k|r) \triangleq p_k$. For each GUI component class, a detection output comprised of the confidence value and refined bounding box position is considered as True-Positive if it has an IoU overlap higher than 0.7 with any ground-truth box belonging to the same class, otherwise as False-Positive. Multiple detections at the same ground-truth box were also regarded as False-Positives.

The three principal metrics used for evaluating and comparing the performance are precision, recall and mean Average Precision (mAP). Precision of each class is obtained through dividing the cumulative sum of True-Positive by the total amount of True-Positive and False-Positive for an individual class. Recall of each class is obtained through dividing the cumulative sum of True-Positive by the total number of ground-truth boxes for an individual class. Average Precision (AP) is obtained by approximating the area under the Precision/Recall curve for individual element class and mAP is the mean of AP across all classes.

IoU	Recall	Precision	mAP
0.6	0.65	0.73	0.69
0.7	0.60	0.79	0.66
0.8	0.53	0.84	0.62
0.9	0.36	0.90	0.44

Table 1. Performance with image augmentation

IoU	Recall	Precision	mAP
0.6	0.51	0.68	0.56
0.7	0.44	0.72	0.48
0.8	0.37	0.80	0.43
0.9	0.26	0.87	0.36

Table 2. Performance without image augmentation

6.3 Evaluation with Annotated Runtime UI Screenshots

For the 6,870 annotated runtime GUI screenshots in the testing dataset, the overall performance of our model is 0.76 for precision, 0.62 for recall and 0.51 for mAP. Fig. 10 shows some examples of our GUI component wirification, and each component is highlighted in bounding box with corresponding component type and probability. We can see that our model can accurately detect UI components in different complex UI screenshots. First our model can handle images in different languages. For example, Fig. 10(a) contains image buttons with English, Japanese and Chinese texts within both the main menu and the popup. Our model can detect all of them in the screenshot. Second, our model is robust to different interference, so it can discriminate GUI components from very similar background in Fig. 10(b). Third, our model is capable to find all components even if there are many of them in one screenshot. For example, there are 10 components including image button, seek bar and check box in Fig. 10(c). Our model can accurately detect all of these components.

Apart from the correctly detected UI components, we also manually check the cases for which our model does work very well, and we summarise two reasons for the incorrect cases. First, sometimes our model cannot identify some specially designed UI components like the circle-shape seek bar in Fig. 10(d). Note that our model still detects the seek bar in the screenshot, but the bounding box is inaccurate due to the irregular shape of the seek bar. Some progress bars are also very different such as the circle one, or rectangle one with different colors, leading to the miss of such UI components. Second, although the GUI components obtained from our model do not match the ground truth, they are still correct. In Android UI development, one functionality can often be implemented in many different ways. For example, the developers can use the image button directly, or using image but setting its attribute as *clickable*. These two ways has the same functionality for user interaction and the choice depends on developers' preference. In such cases, our model may classify an GUI component as an image button, while the ground truth marks it as an image.

6.4 Evaluation with App Introduction Screenshots from Google Play

To further check the performance of our model on the app introduction screenshots in Google Play, we randomly sample 100 app introduction screenshots for the experiment. These sampled images come from 99 different apps across 20 categories. As there is no ground truth for app introduction screenshots, we manually annotate GUI components in each GUI screenshot. We recruit two bachelor students in our school who has at least two-years Android development experience. The two annotators label the GUI components in each sampled app introduction screenshots individually. If there is any disagreement with their annotations, the screenshots will be handed to the first author to resolve the disagreements.

Table 1 shows the performance of our model trained with image augmentation, while Table 2 shows the results without image augmentation. We can see that with the increase of the IoU, the Recall and mAP are decreasing, while the precision is increasing significantly. As we have a large number of app introduction screenshots crawled from Google Play, we are not sensitive

Fig. 11. Results comparison between the model trained with or without image augmentation

to recall because even not a very high recall can still produce a large-scale design gallery of GUI components. But the low precision may include many false-positive UI components which will negatively influence the user experience with our design gallery. Therefore, when we are making a trade off between precision and recall, we emphasize more on the precision in this work. Based on our experiment results, we take the IoU value as the 0.8 so that the precision is high enough while the recall is not very low. Within this setting, our model without image augmentation can reach 0.37, 0.80, 0.43 for recall, precision and mAP, respectively. Compared with it, our model with image augmentation can improve recall, precision and mAP by 43.2%, 5%, and 44.2%, respectively.

Fig. 11 shows some comparison examples with and without image augmentation. Compared with the model trained without image augmentation, the model trained with image augmentation can reliably detects the majority of the GUI components within the app introduction screenshots crawled from Google Play.

7 INFORMAL FEEDBACK FROM DESIGNERS

To understand how the Gallery D.C. would be useful in a real design context, we conducted informal interviews with 7 professional designers.

- D1: Visual designer from Google for 1 year. His focus is on illustration design for various mobile apps.
- D2: Interaction designer/product designer from Volkswagen-Mobvoi Joint Venture, with 10-years working experience. His work mainly focuses on the smart devices.
- D3: Visual designer from TAL education for 5 years of experiences. His focus is on the mobile app UI design.
- D4: Designer from Facebook for 9 years with the focuses on mobile advertising user experience design.
- D5: Visual designer, 13 years of working experience, now working at Ali, responsible for international product design and design innovation.
- D6: Interaction designer from Google. She has 4 years of interaction design experience.
- D7: Visual designer from Huawei with 10 years of experience. Mainly focus on mobile system design.

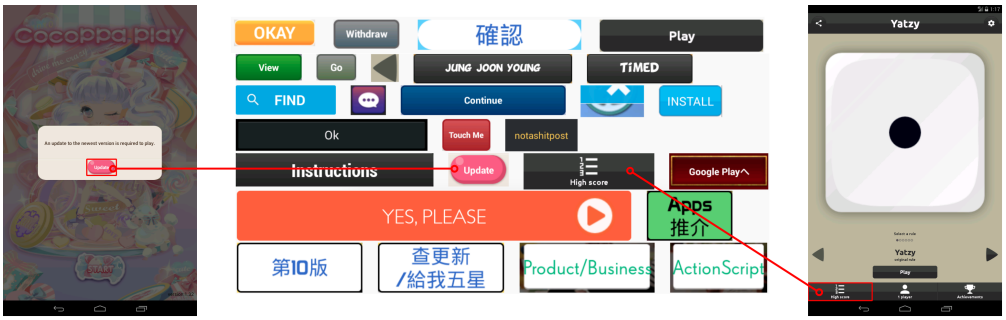


Fig. 12. Two attracting game buttons

For D1 to D5, We demonstrated our Gallery D.C. website and collected general feedback on how our site may be useful for their own app design tasks. For D6 and D7, in additional to general feedback, we also ask them to use our site for at least 20 minutes, and then provide examples in which the system can support their design tasks. Overall, all designers responded positively to our Gallery D.C.. D6 and D7 in particular provide concrete examples of how our system can help their work. All of them also provided suggestions on how to improve the tool. Below, we first provide an overview of the general feedback from D1-D5, then list the detailed tasks highlighted by D6 and D7 on how our system can help with their work.

D1 praised our site for comparing fine-grained design options like the percentage of different colors and size distribution. D2 also likes the summarization and contrast of different design styles. D3 confirmed similar experience and mentions that small companies tend to follow the main-steam design style in the market, while big companies want to have their own unique design style to be distinctive. D4 emphasized the importance of intuitive sense of good designs, and mentioned that our site could help locate the design by such intuitive sense. D5 hoped that our site could play as a bridge between interaction designer and visual designer for enhancing their communications.

7.1 Task 1: Inspirational Search

D6 described a particular use case in which our website can be helpful. In her last job of designing a game app, she needs to decide on style of the buttons that can fit the game she wants to design. She found the component search function particularly useful for looking for inspirations. Using our website, she searched and selected the button component type and set up the app category as *game_casual* in the filter. Within the different buttons, she found two of them attractive, as seen in Fig 12. One button is of pink color with reflective bubble which deemed to attract young girls. Since the target audience of the game app is for young girls, she selected this design and put into her design notebook for reference.

She also find the other button with the ranking-list icon interesting (Fig 12). It is not only straightforward to show the button semantic, but also reminds her to add the ranking mechanism into the game. After further checking the whole screenshots and even other screenshots within this app, she also understands the position of this button in the current screen, and other functionality which can also be embedded into her own app. She found this process of initially using bottom-up search with components, and then gradually identify and expand the design style and concept by checking the design of apps that contains interesting components a promising alternative strategy she would like to adopt into her design practices.

Fig. 13. Buttons from social apps and corresponding demographics

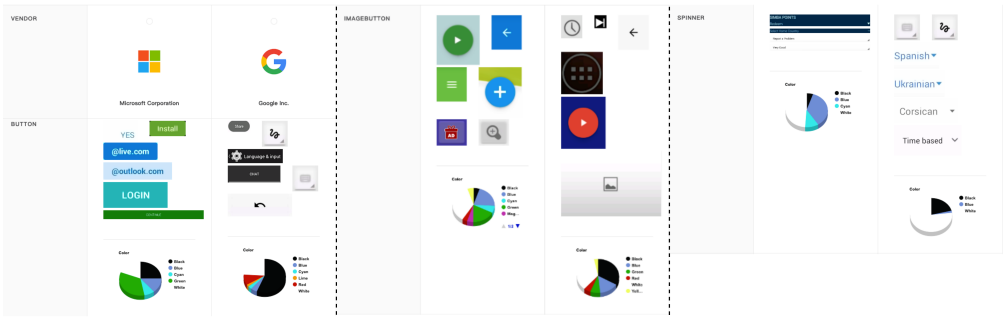


Fig. 14. Comparison of components between Google and Microsoft

7.2 Task 2: Understand Design Demographics

D6 also provides an example of how to use our website to help the design of a social app. In Gallery D.C., she searched the GUI components by social application category. Apart from the buttons as seen in Fig 13, it also showed the color demographics and size distribution on the right side. She discovered that most buttons within the social-medial apps are rather flat and wide as compared with other categories. After recalling her own experience in using social apps and searching the web, she further validated this assumption that the wide and big buttons can easily catch audience's attention, especially for young users who extensively use that kind of app. To D6, the summarization of design features of GUI components is a powerful approach for her to understand the design trend and practices.

7.3 Task 3: Design "Comparison Shopping"

D7 was working on creating the UI component design system [3, 10] which is a set of standards for design and code along with components. Such UI component design system enforces visual and interaction consistency of different products within the company for delivering better and familiar user experiences. To get an understanding of the design style of different companies, he adopted the comparison functionality provided in our Gallery D.C.. He selects two big companies (Google and Microsoft) for detailed comparison.

Our tool returns a list of different components from these two companies as seen in Fig 14. By contrasting these components, he clearly discovered that most components in Microsoft are of the right-angle rectangle. Instead, Google prefers to use white, black and gray color, and Google also embed the shadowing effects to their components. With those features displayed, he further thought about how to distinguish his design system from the current ones.

7.4 Areas for Improvement

Designers point out various areas in which our gallery can be improved with. D6, D7 hope us to further remove some low-quality data within our site as they found that some UI designs returned

by the search do not look professional, hence no reference value. Although we have filtered out some low-quality apps according to their rating score and installation number, we may leverage the crowdsource power to further improve our data quality in the future. Several designers (D1, D2, D3) wonder whether the component detection model could extend to other platforms. For example, their design for desktop, web, or IOS interfaces could benefit from this type of model. We believe our model could help them in this case as it would not be difficult to extend to other platforms once we obtain enough data for the training.

8 RELATED WORK

There are numerous mobile UI design kits available online. Some of them have rather comprehensive mobile app design templates collected from Internet, for example Pinterest [6]. A main drawback of these websites is that the design templates are not specifically categorized and cannot be easily searched. There are also websites for mobile UI designers that provide detailed categories of UI designs, such as Inspired UI [5], Pttrns [7]. They provide many categories of design templates in terms of the whole GUI design, but they do not provide the design resources at the GUI complement level. Some websites provide downloadable and editable GUI components. A typical example is the website Axure Widgets [2]. However, many such websites are not free to use. They are not suitable if somebody just wants to get some design inspirations. Another type of websites, such as Up Labs [3], provide free GUI components, but the UI components are organized by individual providers and it is very hard to find the desired UI design by component metadata. Furthermore, many online design kits have only very limited numbers of UI designs (at most several hundreds). In comparison, our Gallery D.C. contains a large number of GUI components (together with the original whole GUI designs) from over 130,000 mobile applications.

Apart from creating reusable design templates, designers can also share their GUI design artworks on online crowdsourcing platforms such as Dribbble and GraphicBurger. The shared GUI designs are at the whole GUI level, and are tagged by the authors to facilitate the search. Our work exploits another type of invisible crowdsourcing design resources, i.e., app introduction GUI screenshots in the application market which are published by the application developers to demonstrate the applications' important features and UIs. Although these GUI screenshots are not created for the design sharing purpose, we convert them into a large-scale GUI design gallery using data mining techniques. And our gallery supports several advanced design search and knowledge discovery features beyond content sharing and curation of existing online crowdsourcing platforms. Rico [18] uses reverse-engineering and crowdsourcing to collect a large dataset of app runtime GUI screenshots. Based on the static analysis, StoryDroid [17] can more efficiently collect the app GUI screenshots than Rico. Webzeitgeist [23] crawls a large dataset of web page screenshots and corresponding HTML code. They also support some data-driven design applications, but these two works use only runtime GUI screenshots. Compared with Rico, we provide more fine-grained GUI design i.e., GUI components, and also provide a practical tool for supporting the UI component search. Different from Webzeitgeist which mainly focus on the website design, our work is concerned with mobile UI design and adopt more advance deep learning method for decomposing the GUI screenshots into separated components.

In detail, we use app introduction screenshots as the main design resources and collect runtime GUI screenshots to train an object detection model to extract GUI components from app introduction screenshots. To the best of our knowledge, our work is the first to build a large-scale GUI component design gallery using app introduction GUI screenshots and deep learning based computer vision techniques.

Deep learning has been applied to GUI design images in the work of Chen et al. [13]. In that work, an input GUI design image is automatically translated into a skeleton of Android GUI components.

The underling technique is a neural machine translator consisting of a CNN for extracting image features, a RNN encoder for encoding the spatial information, and a RNN decoder for generating the component skeleton. Different from this work, our goal is to decompose an input GUI screenshot into a set of GUI components in the screenshot, and our model relies the object-detection model Faster RCNN [30]. A similar work to ours is the mobile UI reverse-engineering work by Nguyn et al. [28]. But their technique is based on traditional computer vision techniques like edge detection and optical character recognition (OCR) to detect GUI components in an UI screenshot. The method proposed by Moran et al. [26] combines traditional computer vision techniques for GUI component detection and deep-learning based method for GUI code generation. These existing works focus on code generation for an input GUI design. That is, they help with the transition from GUI design to GUI implementation. In contrast, our work focus on facilitating GUI design search and knowledge discovery by providing a large-scale gallery of real-world application GUI components.

There are a few remotely related works in the domain of extracting information from mobile app GUI. For example, the work by Jo and Jung [21] detects logos of mobile phone applications. They search many websites and store the logos and their corresponding website names into a database. In this way, they achieve the goal of "smart learning" of logo design, which is similar to the goal of our design gallery. Other studies related to UI design focus on UI design patterns, such as the study by Meier et al. [25] in which the authors find the relationship between location search patterns and user requirements and study by Swearngin et al [32] for converting the mobile UI screenshots into editable files in Photoshop by using image processing method. Many studies are targeting at searching the GUI for helping software GUI designers and developers [11, 29, 36]. Reiss [29] parses developers' sketch into structured queries to search related UIs of Java-based desktop software in the database. GUIfetch [11] customizes Reiss's method [29] into Android app UI search by considering the transitions between UIs. Similar to Reiss's work [29], Zheng et al [36] parse the DOM tree of user-created rough website to locate similar well-designed website by measuring tree distance. Different from these works, the data granularity of our work is more fine-grained i.e., considering the GUI component search rather than searching the overall GUI screenshot. This is the first work considering the GUI component as the basic unit for inspiring mobile GUI designers.

9 IMPLICATIONS

We discuss some implications of our work on designers, design sharing platforms and the research community.

On designers: GUI design is a very dynamic and creative domain. Designers have to continually learn from others. A common practice nowadays is to learn from online resources (e.g., design kits or blogs) shared by famous designers. Our work identifies a new gold mine of design resources, i.e., app introduction GUI screenshots of real-world applications in the application market. This new gold mine of design resources, once made easily accessible to designers, can very well complement existing online resources, especially "see designs in real life".

On design sharing mechanisms: Our design gallery creates a new way of design sharing in which design authors do not intentionally share their GUI designs but design consumers can still easily access these GUI designs in a well curated way. The bridge connecting the two sides relies on data mining techniques. And compared with the face-to-face design interaction or brainstorming which is limited to several designers in the physical world, our tool based on the large-scale invisible crowdsourcing data enables boarder collaboration indirectly with thousands of designers who craft the design of the world-wide popular mobile apps. Furthermore, our design gallery complements existing design sharing platforms in two aspects. First, unlike existing design sharing mechanisms which keep the whole GUI designs and the GUI component designs separated, our design gallery naturally links the two granularities of design information. Linking whole-component design

granularities gives designers more flexibility to access design knowledge. Second, unlike existing design sharing platforms which focus mainly on content hosting and management [12, 16], our design gallery is equipped with advanced design search and knowledge discovery features which help designers explore multi-faceted design space and distill higher-order of design knowledge.

On research community: With the advent of Web 2.0, user generated content has become an important knowledge source, for example Wikipedia in general domain, Stack Overflow in computing domain, and Dribbble in design domain. A great deal of research has been done to support the creation, sharing and curation of user generated content. In addition to such intentional crowdsourcing, there are many invisible crowdsourcing resources, such as app introduction GUI screenshots of real-world applications in the application market in this work. Invisible crowdsourcing resources are not created for the purpose of content sharing, so they are much more difficult to collect, curate and exploit. Our work demonstrates a successful case of collecting, curating and exploiting invisible crowdsourcing resources, turning invisibles into explicit knowledge. This research direction is promising and deserve more attention from the community.

10 CONCLUSION AND FUTURE WORK

In this paper, we present a deep-learning based approach for harvesting invisible crowdsourcing GUI design resources in the application market, which allows us to build a large scale GUI component design gallery. The core techniques to turn invisible crowdsourcing GUI design resources into a design gallery includes a reverse-engineering technique to collect app runtime GUI screenshots and an object-detection deep learning model to decompose a whole GUI design into a set of GUI components. Once those invisible design resources are made explicit in our design gallery, they complement existing design sharing platforms by exposing designer to not only design creativity and aesthetics but also design practicality. Furthermore, our design gallery contains both whole GUI designs and GUI components which give designers more flexibility to access design knowledge at different granularities. Last but not least, our design gallery supports multi-faceted design search, design demographics summarization, and design comparison shopping. These design applications can be incorporated into existing design sharing platforms to extend them beyond content sharing and curation.

In the future, we will keep improving our work according to the feedback from designers. First, we will leverage the crowdsourcing method to filter out apps with low-quality UI design. Users will get professional design as the query results for better inspirations. Second, we will improve our tool with better user experience including more search and navigation features. For example, our tool may take the component image as the query, and not only return similar components, but also other-type components whose design style can fit into the query's. Third, we will extend our approach into other platforms including website, IOS, or other IOT (Internet of Things) devices for supporting broader UI design.

ACKNOWLEDGMENTS

We appreciate assistance from Chow Wei Jun Bernard in carrying out some experiments, and Koh Hong Da, Yuyang Wang, Siyuan Jiang for the demo website construction. We also acknowledge the GPU support from Nvidia for accelerating the experiments. This project is partially supported by the FIT ECR seed grant and Multi-disciplinary seed grant in Monash University.

REFERENCES

[1] 2013. Elven iPhone App UI Kit. https://graphicburger.com/elven-iphone-app-ui-kit/. Accessed: 2018-04-25.
[2] 2018. Axure Widgets. https://axurewidgets.com/download-axure-widgets/. Accessed 2018.
[3] 2018. The Global Network For Creatives GET STARTED. https://www.uplabs.com. Accessed: 2018-04-25.

[4] 2018. Google Play. https://play.google.com/store/apps. Accessed: 2018-04-25.
[5] 2018. Inspired-ui. http://inspired-ui.com/. Accessed 2018.
[6] 2018. Mobile app design. https://www.pinterest.com.au/timoa/mobile-ui-photos/. Accessed 2018.
[7] 2018. Pttrns. https://pttrns.com/. Accessed 2018.
[8] 2018. UI Automator. https://developer.android.com/training/testing/ui-automator. Accessed: 2018-04-25.
[9] Andrea Asperti and Claudio Mastronardo. 2017. The Effectiveness of Data Augmentation for Detection of Gastrointestinal Diseases from Endoscopical Images. *arXiv preprint arXiv:1712.03689* (2017).
[10] Young-Min Baek and Doo-Hwan Bae. 2016. Automated Model-based Android GUI Testing Using Multi-level GUI Comparison Criteria. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)*. ACM, New York, NY, USA, 238–249. https://doi.org/10.1145/2970276.2970313
[11] Farnaz Behrang, Steven P Reiss, and Alessandro Orso. 2018. GUIfetch: supporting app design and development through GUI search. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*. ACM, 236–246.
[12] Chunyang Chen, Xi Chen, Jiamou Sun, Zhenchang Xing, and Guoqiang Li. 2018. Data-driven proactive policy assurance of post quality in community q&a sites. *Proceedings of the ACM on human-computer interaction* 2, CSCW (2018), 33.
[13] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. 2018. From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation. *The 40th International Conference on Software Engineering (ICSE)* (2018).
[14] Chunyang Chen and Zhenchang Xing. 2016. Mining technology landscape from stack overflow. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 14.
[15] Chunyang Chen, Zhenchang Xing, and Lei Han. 2016. Techland: Assisting technology landscape inquiries with insights from stack overflow. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 356–366.
[16] Chunyang Chen, Zhenchang Xing, and Yang Liu. 2017. By the community & for the community: a deep learning approach to assist collaborative editing in q&a sites. *Proceedings of the ACM on Human-Computer Interaction* 1, CSCW (2017), 32.
[17] Sen Chen, Lingling Fan, Chunyang Chen, Ting Su, Wenhe Li, Yang Liu, and Lihua Xu. 2019. Storydroid: Automated generation of storyboard for Android apps. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 596–607.
[18] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 845–854.
[19] Ross Girshick. 2015. Fast r-cnn. *arXiv preprint arXiv:1504.08083* (2015).
[20] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2980–2988.
[21] Insoon Jo and Im Y Jung. 2016. Smart learning of logo detection for mobile phone applications. *Multimedia Tools and Applications; Dordrecht* 13211-13233 (2016).
[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[23] Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R Klemmer, and Jerry O Talton. 2013. Webzeitgeist: design mining the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 3083–3092.
[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[25] Sebastian Meier, Frank Heidmann, and Andreas Thom. 2014. A comparison of location search UI patterns on mobile devices. *MobileHCI '14* 465-470 (2014).
[26] Kevin Moran and Carlos Bernal-Cardenas. 2018. Machine Learning-Based Prototyping of. *SOFTWARE ENGINEERING JOURNAL*, (2018).
[27] Bashir Muhammad and Syed Abd Rahman Abu-Bakar. 2015. A hybrid skin color detection using HSV and YCgCr color space for face detection. In *Signal and Image Processing Applications (ICSIPA), 2015 IEEE International Conference on*. IEEE, 95–98.
[28] Tuan Anh Nguyen and Christoph Csallner. 2015. Reverse Engineering Mobile Application. *International Conference on Automated Software Engineering* (2015).
[29] Steven P Reiss, Yun Miao, and Qi Xin. 2018. Seeking the user interface. *Automated Software Engineering* 25, 1 (2018), 157–193.
[30] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99.

[31] Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, and Zhendong Su. 2017. Guided, stochastic model-based GUI testing of Android apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 245–256.

[32] Amanda Swearngin, Mira Dontcheva, Wilmot Li, Joel Brandt, Morgan Dixon, and Andrew J Ko. 2018. Rewire: Interface Design Assistance from Examples. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 504.

[33] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.

[34] Georg Wimmer, Andreas Uhl, and Andreas Vecsei. 2017. Evaluation of domain specific data augmentation techniques for the classification of celiac disease using endoscopic imagery. In *Multimedia Signal Processing (MMSP), 2017 IEEE 19th International Workshop on*. IEEE, 1–6.

[35] Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. 2016. Understanding data augmentation for classification: when to warp?. In *Digital Image Computing: Techniques and Applications (DICTA), 2016 International Conference on*. IEEE, 1–6.

[36] Shuyu Zheng, Ziniu Hu, and Yun Ma. 2019. FaceOff: Assisting the Manifestation Design of Web Graphical User Interface. (2019).

Received April 2019; revised June 2019; accepted August 2019