Design Composition Critique

Sidong Feng

A thesis submitted for the degree of
Bachelor of Software Engineering
The Australian National University

October 2019

Except where otherwise indicated, this thesis is my own original work.

Sidong Feng
24 October 2019

to my teachers, parents and friends

Acknowledgments

I am sincerely grateful to Dr. Zhenchang Xing and Dr. Chunyang Chen for their support of my study and research, for their illuminating views on issues related to the project.

Besides my supervisors, I express my warm thanks to my parent for their support and encouragement.

My sincere gratitude also goes to everyone who gave me precious support during the project. I would also like to express my special thanks to my friend Ms. Biwei Cao for her assistance on grammar. I would also like to thank my project external guide Ms. Yaxian Wang from the Google Inc. and Ms. Yaxian Wang from the Leju Inc. for their insightful comments which incented me to improve my research skills in various fields.

Abstract

Efficiently searching for high quality UI is an immensely important way for designers to gain inspiration, yet existing design sharing platforms can not satisfy designers' requirement. First, designers look for not only the creative concept of GUI designs, but also the practical use of certain GUI designs in real applications. Second, designers want to see not only the overall designs but also the detailed design of the GUI components. In this project, we exploit millions of art designs and real-world applications to build a comprehensive search system, Design Gallery (http://mui-collection.herokuapp.com/). On the one hand, we support the art design search for advanced tags through UI semantics and deep neural network methods. On the other hand, we support a multi-faceted search of practical components that are drilled down using reverse-engineering and computer vision techniques. Through a process of invisible crowdsourcing, Design Gallery provides a platform for designers to collect, analyze, search and reuse GUI and GUI components on a massive scale. Design Gallery is responded positively via informal evaluation with professional designers and proves of fact that it offers additional support for design sharing and knowledge discovery beyond existing platforms.

Contents

List of Figures

List of Tables

Introduction

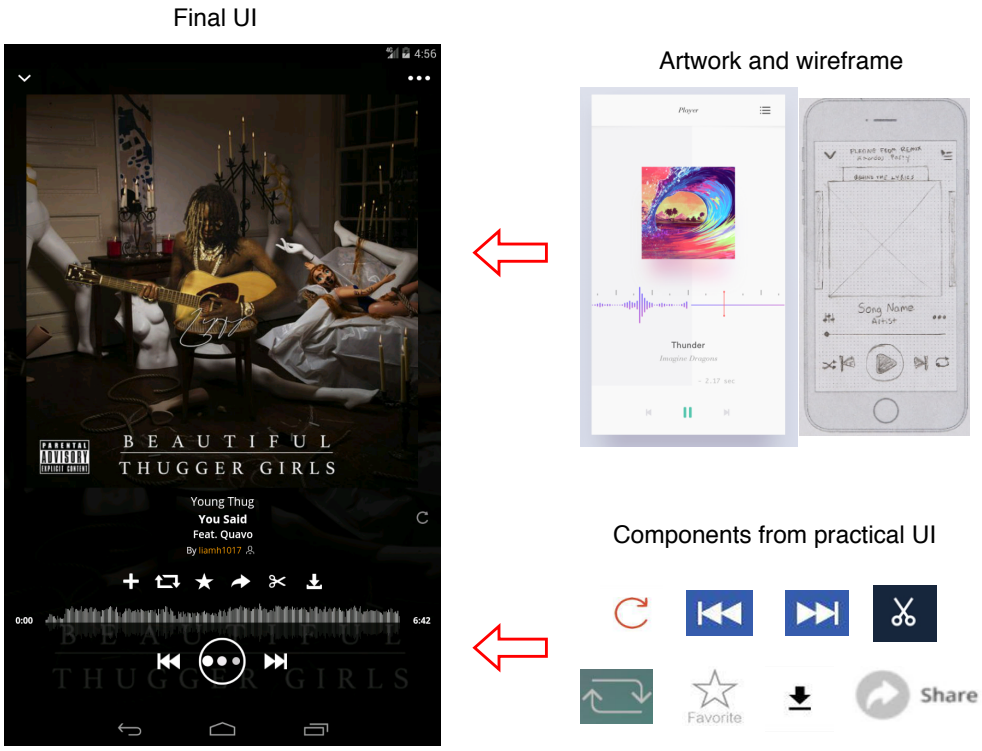


Figure 1.1: Examples of UI design consists of both whole UI in art work and components in practical work.

1.1 Introduction

User Interface (UI) is ubiquitous in almost all modern desktop software, mobile applications and online websites. It provides a visual bridge between a software application and end-users, through which they can interact with each other. A good UI design makes an application easy, practical and efficient to use, which significantly affects the success of the application and the loyalty of its users [Jansen, 1998;

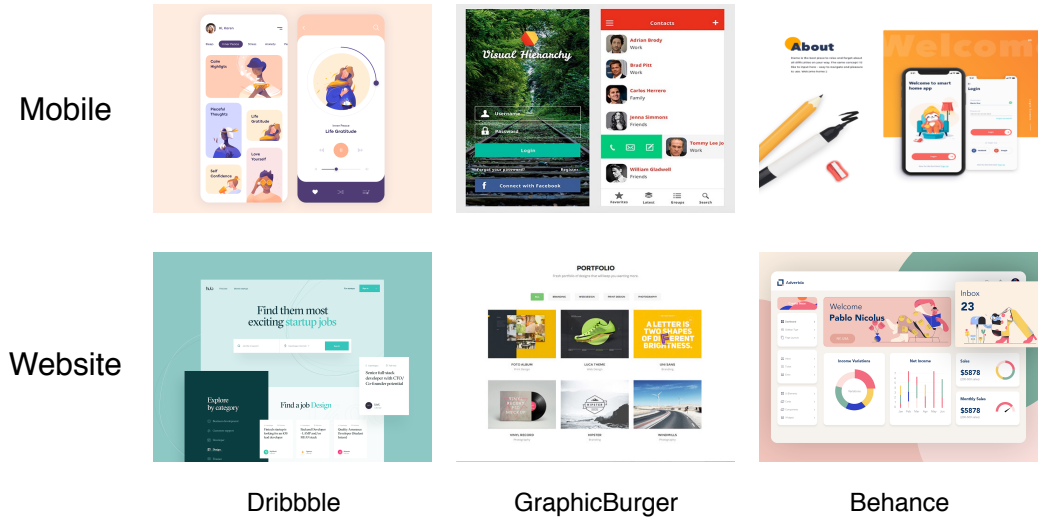


Figure 1.2: Examples of GUI design artwork shared on Dribbble[Dribbble, 2010], GraphicBurger[GraphicBurger, 2013] and Behance[Behance, 2005]

binti Ayob et al., 2009; Anderson et al., 2010]. For example, computer users view Apple's Macintosh system as having better UI than Windows system, therefore their positive views almost double that of Windows users, leading to 20% more brand loyalty [Winograd, 1995].

A good UI design is difficult and time-consuming [Shneiderman and Plaisant, 2010]. One way to design a good UI is to learn from other professional designs, including aesthetics and practicality. Artwork follows the aesthetic supremacy, which allows designers to gain inspiration and creativity [Fitzmaurice et al., 1999]. However, the art work does not operate on the level of functionality. In contrast, a practical UI design follows many design principles and constraints, such as fluent interactivity, universal usability, clear readability, aesthetic appearance, consistent styles [Ryan and Gonsalves, 2005; Galitz, 2007; Clifton, 2015; web, 2017]. Thus, designers need to learn from both the aesthetics in art work and the practicality in real work. Furthermore, to facilitate their design process, designers also reuse the design such as the whole UI composition and component kits. The whole UI provides the consistency and composition in design, including size consistency, color consistency, components composition, etc. Independent to the design logic or schema, component kits can be reused as a pure resource. For example, in Figure 1.1, the final UI design distills the wireframe from artwork, and also reuses the components in practical work. To assist designers in terms of learning and reusing, we offer a design gallery with two searching interfaces, one is the whole UI design in artwork, and the other one is the component-oriented design in practical work.

With the advent of Web 2.0, art designs are widely shared among online communities. For example, Figure 1.2 shows some GUI designs shared on Dribbble[Dribbble, 2010], GraphicBurger[GraphicBurger, 2013] and Behance[Behance, 2005]. Although these design sharing websites have enormous amounts of designs, it is still diffi-

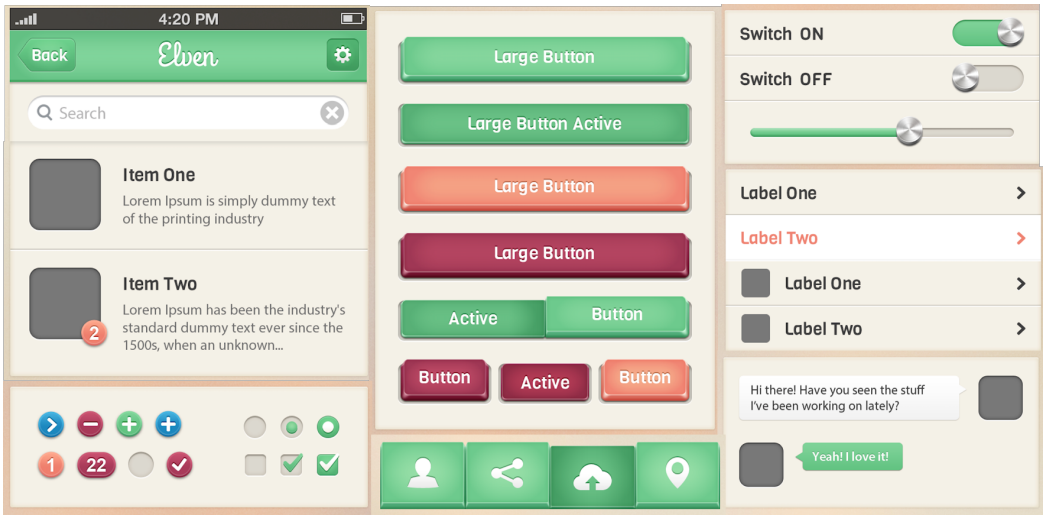


Figure 1.3: Examples of design kits of GUI components [GraphicBurger, 2013]

cult to find desirable designs efficiently [Bernal-Cardenas et al., 2019]. This is because the search interface lacks of the ability to translate graphical designs into textual description (i.e., tag). For example, when searching a specific tag (i.e., food) in Dribbble [Dribbble, 2010], the search interface retrieves the designs only if the tags annotated by designers is fully matched with the query, but not the concept of graphical designs. This kind of searching technique is also denoted as tagging based search [Dasdan, 2011]. Since the tagging based search targets on pre-described tags by designer, there are two problems in practice. First, human can easily make mistakes on vocabulary such as abbreviations and synonyms. For example, if designer annotates the design with "visualise" tag, it is unsearchable by "visualize". Second, human can easily omit the tags. For example, the designer mainly works in a game company rarely annotate statistical tag such as "chart", even though the graphical design includes a chart diagram. These two problems can severely degrade the performance of the tagging based search. We develop two approaches to solve this problem. First, to solve the abbreviations and synonyms, we construct a structured vocabulary of UI semantics using 61,700 UI designs collected from Dribbble. Second, to complement missing tags for UIs, we employ a hybrid neural network that leverages both the visual and textual information. Since single design can be annotated by several tags, we define our problem as a multi-label problem. To solve this problem, we adopt a binary classification for each tag (i.e., the design can be annotated by this tag or not). We will further describe our procedure in Section 2.2. To present our model visually in terms of helping designers gain inspirations, we build a design gallery(http://mui-collection.herokuapp.com/test/). The designers can perform search by keywords and categories. Furthermore, a ranking system is applied to bring the most relevant designs to designers.

Apart from the art designs, GUI designs can also be shared in the form of reusable design templates or design kits [GraphicBurger, 2013; Axure, 2018; Inspired-ui, 2018;

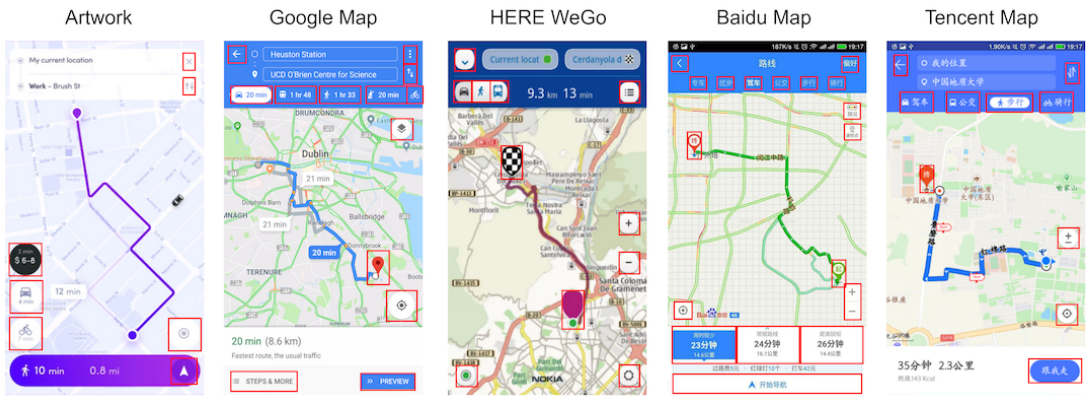


Figure 1.4: Examples of GUI design artwork versus real application GUIs

Pttrns, 2018; Uplab, 2018] (see Figure 1.3 for examples). Through design sharing, the designers can gain feedback from the community and boost their portfolio. However, 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. GUI design artwork and real application GUIs may have substantial differences. Figure 1.4 shows an example of map application GUIs. We can see that the map-GUI artwork does include the core GUI components of a map application, but it is much simpler than the GUIs of real world map applications. On the contrary, the complexity of the four real-world map applications GUIs is roughly the same. This suggests that the information needs of map users in the real world go beyond those core needs supported in a simple design artwork. It is fine to ignore these additional information needs in the design work. However, if the real-world applications leave these needs unattended, they risk losing valuable users to their competitors. 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 Figure 1.3), 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 tagging-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. To confront the fundamental limitation of *design practicality*, we exploit Android application market (i.e., Google Play [Google, 2019]). 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. To address *design granularity*, we decompose the components in automatically collected 68,702 GUI screenshots from application in Google Play. However, collecting and inspecting real-application GUIs components in an ad-hoc fashion cannot effectively exploit this gold mine of GUI design resources. To obtain more GUI components from high-quality GUI designs such as Rico [Deka et al., 2017], we use the GUI design dataset obtained through automated GUI exploration to train a deep-learning object detection model (Faster RCNN [Ren et al., 2015]) 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. In addition to the component metadata, we also detect the primary color of GUI components in HSV color space [Muhammad and Abu-Bakar, 2015], and collect app metadata such as app category and download times. To achieve *design knowledge discovery*, we build a design gallery. Based on the GUI components in the collected application runtime screenshots and Rico, we build a web application design gallery (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 compares the GUI designs from 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.

Our contributions can be summarized below:

- We identify two core implicit demands for designers to design a high quality systematic UI design. First, understand the design knowledge in both artistic and practical UI designs (i.e., design trends). Second, reuse available designs including whole UI and component.
- Whole UI Search
  - We systematically investigate the UI design tagging problem and contributed a working artefact that can assist designers with the UI search.
  - We construct a vocabulary for UI design semantics based on the tags for the large-scale UI design from Dribbble.
  - Based on the vocabulary, we develop a novel deep-learning based method for specifically recommending semantic tags to the existing design, and it outperforms traditional methods.
- Component-Oriented Search
  - 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 GUI screenshots into a large-scale GUI component design gallery, which contains design creativity and knowledge of millions of application designers and developers.

- Our design gallery enables crowdsourcing of GUI design resources and new ways of design sharing and design knowledge discovery beyond content sharing.

1.2 Motivation

An intuitive, pleasant-to-use Graphical User Interface (GUI) is crucial for the success of a mobile app in the competitive market. Designing a good UI must follow many design principles and styles, and at the same time, requires designers' creativity and consideration of users' needs. Unfortunately, this design work often awaits just a few designers in a company. For example, the ratio of software developers to UI designers at Microsoft was 50:1, and this ratio was already better than many other companies out there [Hong, 2011]. Due to the shortage of UI designers, software developers have to fill in the gap. A survey [web, 2015] of 5,778 mobile developers indicates that 51% reported they work on UI design tasks more than development tasks.

In the past decades, the research community tries to fill this need. The research [Subramanya and Yi, 2007; Yang and Klemmer, 2009; Zen and Vanderdonckt, 2014; Silvennoinen et al., 2014] focus on main issues of "the perceptions of aesthetics-usability relationships in UI design". They have indicated that while the usability is of high importance, aesthetics is a crucial factor in determining the GUI quality as it affects the UI of the product in many ways. First, it attracts users. As a matter of fact, users easily feel disappointing on bad designs and try a new app. In contrast, a good UI design can improve the users trust, consequently, saving users. Second, users have more tolerances with usability issues in a good UI design. Studies indicate that rate hinges on the design. A visually appealing design normally gets high votes as users feel pleasant in using them [Norman, 2004]. However, researches primarily contribute to the discussion of evaluating and building UI design, including design principle [Galitz, 2007; Fu, 2010], design evaluation metric [Zen and Vanderdonckt, 2014] and interface building tools [Borning and Duisberg, 1986; Szekely et al., 1993], while only a few researches provide a resource for designers to get inspirations and reuse in real work. Design sharing is a routine activity through which design knowledge, creativity and aesthetics is exchanged among designers. Therefore, we build a design gallery that give designers more flexibility to access design knowledge at the level of art and practice, at the level of whole UI and component-oriented.

1.3 Thesis Outline

The report is structured as follows:

- **Chapter 2** introduces an advanced searching based on the whole UI art designs.
- **Chapter 3** proposes an advanced knowledge discovery based on the component-oriented practical designs.
- **Chapter 4** draws a conclusion for the project and discusses the future works.

Chapter 2

Whole UI Search

Despite the enormous amount of UI designs existed online, it is still difficult for designers to efficiently find specific examples to inspire their design work, due to the miss-matching problem between search queries and UI design images. To facilitate better matching, design sharing sites like Dribbble [Dribbble, 2010] ask users to provide tags when uploading UI examples. However, designers often do not know how to properly tag a design, resulting in unclear, incomplete, and inconsistent tags for uploaded examples which impede retrieval, according to the interview with four designers. Based on the deep neural network, we introduce an automatic approach for encoding both the visual and textual information to recover the missing tags for UI examples so that they can be more easily found by text queries. Our deep learning model is trained using a vocabulary of UI semantics, constructed through an iterative open coding process of existing tags. We achieve 82.72% accuracy in the tag prediction. This project also presents a design gallery (http://mui-collection.herokuapp.com/test/) to assist designers to efficiently find what they want. Through a simulation test of 4 queries, our system on average return much more results as compared with Dribble, leading to better relatedness, diversity and satisfactoriness. This chapter is structured as follows:

1. **Section 2.1** investigates the related work including searching algorithm and tag predictions.

2. **Section 2.2** proposes a workflow of our searching system including data collection, UI semantics construction and tags prediction.

3. **Section 2.3** analyses the performance of our proposed model (Hybrid model) in three aspects, including model performance, baseline comparison and model behaviours.

4. **Section 2.4** introduces the gallery system design and construction, including gallery database, search filter, existing platforms comparison and ranking system.

5. **Section 2.5** understands the usefulness of our model and gallery in a real design context via an informal interview with professional designers. At the end, we discuss the areas for improvement.

2.1 Related Work

GUI design is a creative knowledge work. To assist the UI design, many studies are working on data mining in UI designs such as Alharbi et al. [Alharbi and Yeh, 2015] investigate a UI design pattern, Jahanian et al. [Jahanian et al., 2017b,a] develop a color evolution method, Fu et al. [Fu et al., 2013] and Martin et al. [Martin et al., 2017] discuss the UI-related users' review. To personalize GUI dynamically, Fischer et al. [Fischer et al., 2018] adopt a style transfer algorithm from fine art to GUI. To assist the semantics of UI design, Liu et al. [Liu et al., 2018] annotate the mobile GUI design by following the libraries from Balsamiq and Google's Material Design. To emphasize the reusability of design, Swearngin et al. [Swearngin et al., 2018] propose a image processing method that allows UI screenshots convert into editable files in Photoshop. Therefore, it allows designers to further customize their design. Chen et al. [Chen et al., 2019] develop a program analysis method that takes an app executable file as input and return a UI screenshots storyboard. Therefore, it helps designers to gain inspirations. Previous works are well suited for understanding and simplifying the UI designs. In contrast, our study is to assist designers with searching the online platform.

To facilitate searching techniques, many researches analyse the usefulness of Retrieval-based methods [Reiss et al., 2018; Behrang et al., 2018; Zheng et al., 2019]. To extend the input of searching, Deka et al. [Deka et al., 2017] develop an auto-encoder to support a rough sketch searching. Reiss et al. [Reiss et al., 2018] translate sketch into XML-based vector, and then search matching UIs by measuring vector distance in Java based desktop. Many studies regard Reiss work as a starting point. The transitions between UIs is able to retrieve in the work by GUIfetch et al. [Behrang et al., 2018]. Instead of parsing XML-based vector, Zheng et al. [Zheng et al., 2019] transform the rough website into a DOM tree. All of these works facilitate searching with different inputs including textual tag and visual sketch. In contrast, we refer to search UI designs by using a high-level natural-language semantic tags UI-related association and a hybrid model capturing textual and visual information.

Automatically annotating image (a.k.a automatic image tagging) has emerged as a problem in computer vision and content-based image retrieval [Li and Wang, 2003]. Mori et al. [Mori et al., 1999] map words onto image by measuring co-occurrence of sub-images. Duygulu et al. [Duygulu et al., 2002] propose a Translation Model to label every segmentation. To improve the efficiency on Translation Model, Jeon et al. [Jeon et al., 2003] annotate the image by clustering the segmentation. While the previous works perform well in annotating images, it is expensive on dividing or segmenting images. Furthermore, sub-image or segmentation loses potential information due to elements in the image have an inherent connection. In contrast, our work annotates the image as a whole to capture all information as well as retaining efficiency.

Many research works have denoted it as a multi-class multi-label classification problem. Some of them propose machine learning methods by using manual-craft features, including Gaussian Mixture Models [Wang et al., 2009], Non-negative Ma-

trix Factorization [Kalayeh et al., 2014] and Co-regularization [Chen et al., 2013]. A comprehensive survey of the existing techniques for multi-class multi-label classification is conducted by Wang et al. [Wang et al., 2012]. Previous works achieves state-of-the-art performance, but instead of adopting multi-class multi-label model, we decompose tag recommendation for UI design into an independent binary classification problem for each tag. This is because, there are two fundamental limitations in previous model. First, multi-class task easily aggravates the imbalanced distribution because of the size of data in each class varies [Pedrycz and Chen, 2011]. Second, multi-label task easily suffers from the difficulty of high dimensionality as the dimension is linear to the label size [Wang et al., 2010a]. Different from previous work, we further boost our model using user-provided tags as a joint deep learning on textual and visual information. From the level of image object, all of the works contribute on the natural images such as Corel, Flickr, etc. In contrast, our work focus on aesthetics GUI design comes from human creativity.

2.2 Design and Implementation

Online platforms allow GUI designers to share their design art work in online communities such as Dribbble [Dribbble, 2010] and GraphicBurger [GraphicBurger, 2013]. In our work, we mainly focus on art works from Dribbble. This is due to two reasons. First, Dribbble has enormous amount of UI designs (e.g., 6 million designs). Despite the number of designs, it also contains a wide diversity of designs (e.g., user interface design, logo design, illustration, photography, etc.). Second, design creator is able to annotate the design with 20 tags, which is larger than others. More tags allow our model to capture more textual information. In the following section, we first introduce our dataset, then we discuss the semantic in UI. Finally we propose an end-to-end model to recover missing tags.

2.2.1 Data Collection

To collect image and associated data from Dribbble, we build a web crawler [Heydon and Najork, 1999]. Normally, crawling process is expansive and time consuming [Amalfitano et al., 2011]. To achieve efficiency, we adopt Breath-First search strategy [Najork and Wiener, 2001]. We first collect a queue of URLs as a seed. Then, we crawl all the information from the head URL of queue and we further put the seeds in the URL into the queue as a second stage (see in Figure 2.1).

In Figure 2.2, we show an example of the design work in Dribbble. We can see that a graphical design is attached with information such as title, designer name, a short description, tags annotated by the designer, etc. Consequently, our crawler collects the static graphic design (in png, jpg format) with the corresponding information. Although we are concerned with the UI related design, we still collect the wide diversity of graphical designs including user interface design, logo design, illustration, photography and so on. This is because these designs can be considered

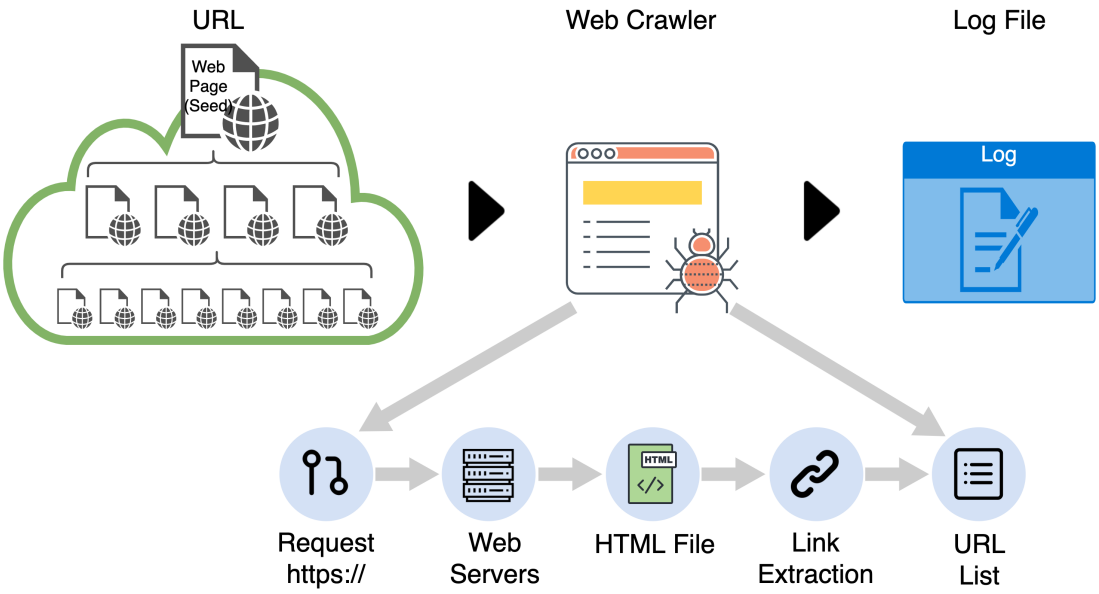


Figure 2.1: An illustration of Breath-First web crawling algorithm [Najork and Wiener, 2001].

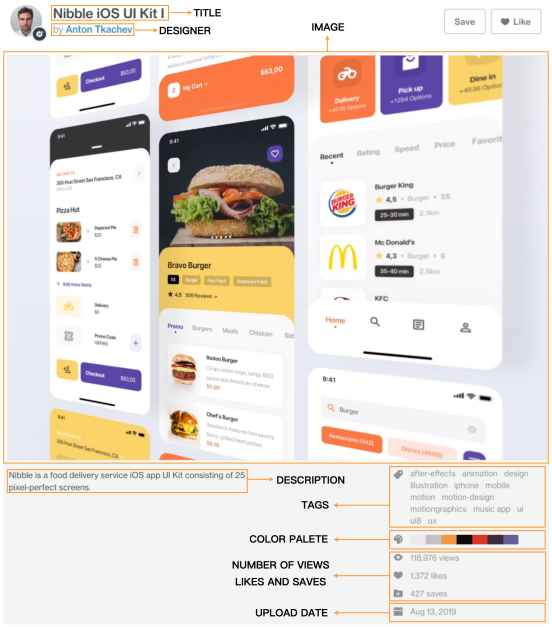


Figure 2.2: A sample design entitled "Nibble iOS UI Kit I" from the design sharing website `www.dribbble.com` of which tags illustrate the two problems with tagging-based search.

as our negative data to train, and we will elaborate this part in Section 2.2.3.1. Although we can determine whether the graphical design is related to UI by using the

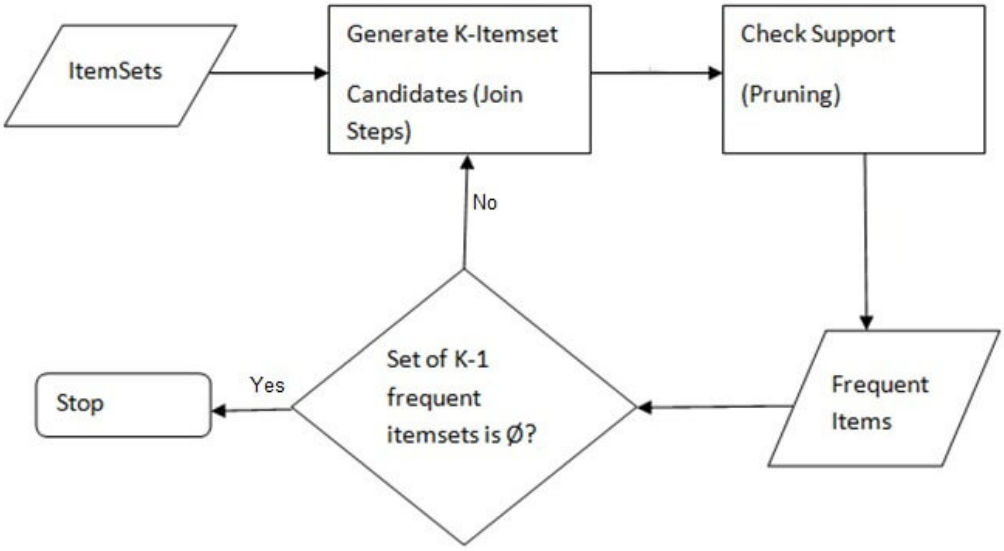


Figure 2.3: A workflow of apriori algorithm [Inokuchi et al., 2000]

inherent "UI" tag annotated by design creator, we still need to explore the potential UI-related tag, such as "user interface", "app design", "website design", etc. We construct a vocabulary to distill semantic relation in Section 2.2.2. Overall, we collect 61,700 UI design with associated metadata.

2.2.2 UI Semantics

To understand the relationship between tags, we first apply an algorithm to distill the semantics. Then, we present a landscape to visualise different communities in the UI tags. Next, we construct a vocabulary table to demonstrate the high-level tag categories. Finally, we carry out an approach to cover abbreviations and synonyms of tags.

2.2.2.1 Overview of UI semantics

As our observation, there are two potential problems on the tags. First, the category should be clear and distinct. For example, "sport" is a commonly used category, therefore, is "fitness" required to be another category as a category as "fitness" relates to "sport" in most case. Therefore, the frequent pairs of "sport" and "fitness" needs to be discovered. Second, the correlation of tags is implicit. For example, in Figure 2.2 "iphone" is one kind of "mobile" phone. However, Dribbble manages the design tags simply as a set of terms. To solve these problems, we apply association rule mining [Agrawal et al., 1994]. It is a widely used method to find patterns and associations in application such as data analysis and marketing.

We totally collect 1,577,695 tags from Dribbble, we adopt aprori algorithm [Inokuchi et al., 2000] to distill the relations of tag semantics because it is devised to operate on a large database. An illustration of aprori algorithm workflow is shown in Figure 2.3.

Design#	Tags
1	iphone, ios, mobile, white
2	sport, iphone, ios, mobile
3	iphone, ios, mobile, grid
4	mobile, iphone
5	iphone, ios, mobile, white

Table 2.1: An usage scenario of tags in UI design.

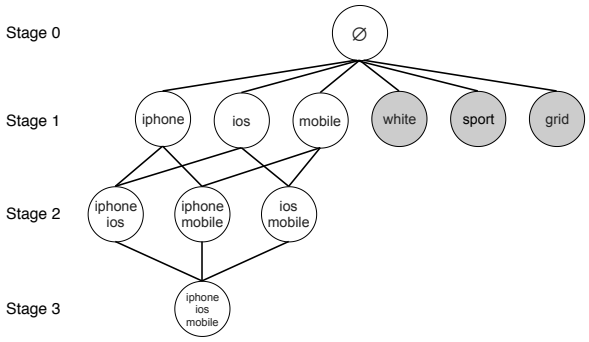


Figure 2.4: An illustration of apriori algorithm in the scenario with t_{sup}=0.75.

We first construct a item set $I = \{I_1, I_2, I_3, .., I_n\}$ which is the item in transactions. Then, we define an itemset (cartesian product) based on stage k (i.e., k = 0, 1, 2, etc.). For example, 2-itemset is a set of $\{I_1 + I_2, I_1 + I_3, I_2 + I_3, ...\}$. Next, we prune the unsatisfied items based on the threshold of Support, which measures the proportion of transactions in which the item appears. We keep generating itemset and pruning until the frequent itemset is found. To gain better insight on how aprori algorithm works in our problem, we assume a scenario in Table 2.1. For example, design#1 is annotated with 4 tags, including "iphone", "ios", "mobile" and "white". First, we construct a basic tree (0-stage: empty node, 1-stage: all the tags in designs). Then, we prune tags like "sport", "white" as their support values are below the minimal threshold t_{sup} (i.e.,$Support\{sport\} = \frac{1}{5} < t_{sup}$) (see in Figure 2.4, the grey nodes denote the pruned nodes). As a result, we obtain the itemset "iphone"+"ios"+"mobile". It means that these set of tags is frequently appear together in the designs, therefore, they have an implicit correlation.

To further discover pair-wise correlation in tags, we calculate confidence which determines how likely item I_1 is tagged when item I_2 is tagged, expressed as $\{I_2 \Rightarrow I_1\}$.

$$\{I_2 \Rightarrow I_1\} = \frac{Support\{I_1, I_2\}}{I_2} \tag{2.1}$$

A low confidence value of $\{I_2 \Rightarrow I_1\}$ means that I_2 is not crucial to determine I_1. Therefore, we set a confidence threshold t_{conf} to filter out the weak correlation. Based on the pair-wise correlation, we construct an undirected graph $G(V, E)$. The tag represents the node V and the edge E represents the pair-wise correlation. Note that the graph is undirected because $\{I_2 \Rightarrow I_1\}$ only indicates the correlations between antecedent and consequent. As a result, we obtain 96 nodes and 202 edges with t_{sup} as 0.002 and t_{conf} as 0.005. To further carry out the community detection [Blondel et al., 2008], with the help of Gephi tool [Bastian et al., 2009], we present a graph in Figure 2.5.

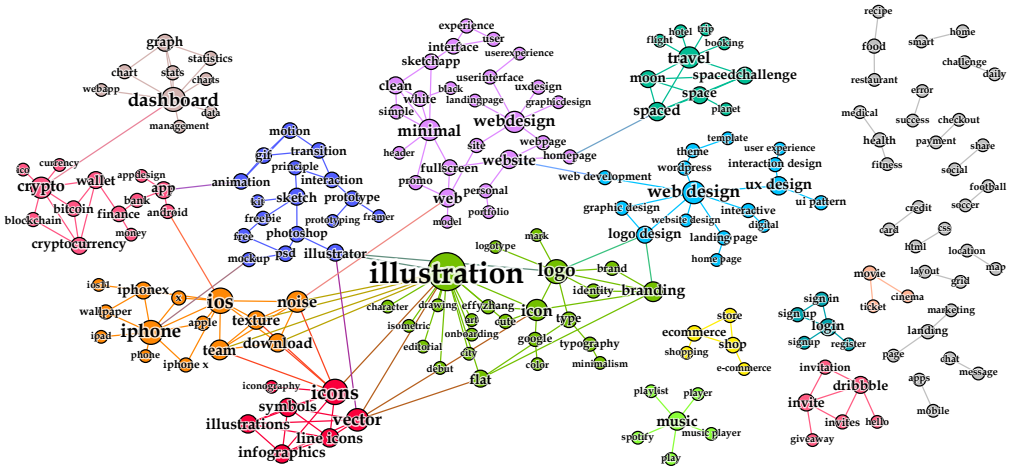


Figure 2.5: The UI-related tag associative graph

2.2.2.2 Vocabulary for UI semantics

From Figure 2.5, we can see that there are many related tags linked together, for example, ("music", "playlist"), ("health", "medical"), ("payment", "checkout"), etc. In order to identify the set of frequently occurred UI tag categories, we adopted a consensus-driven, iterative approach to combine the observed tag landscape generated by our method with existing expert knowledge documented in books and websites such as *Mobile Design Pattern Gallery* [Neil, 2014] and *Google's Material Design* [Material, 2014]. Since the tags cover internal structural information (e.g., layout) and natural information (e.g., color, application category), we manually split the categories. First, we independently coded the categories among these tags. We then meet together and discuss discrepancies. Finally, The categories is defined as long as consensus is reached.

At last, we result in 5 main semantic UI categories: PLATFORM, COLOR, APP FUNCTIONALITY, SCREEN FUNCTIONALITY, SCREEN LAYOUT. Each main category also contains some sub-categories, as seen in Table 2.2. For example, the APP FUNCTIONALITY category contains "Music", "Game", "Sport", and the sub-category "Sport" contains tags "sport", "gym", "workout", etc.

2.2.2.3 Normalize abbreviation and synonyms

During the process of coding the categories of UI semantic, we find that the tag may be written in many *morphological forms* including *abbreviations, synonyms and misspellings* because the designers have very diverse technical and linguistic background. For example, the tag "visualisation" can be written with synonyms such as "visualisation", "visualizations" and misspelling such as "vizualization". The wide presence of *morphological forms* poses a serious challenge to the information retrieval and also the vocabulary construction. For example, the query “ecommerce app” may not find

CATEGORY	ASSOCIATED TAG NAME
PLATFORM	
Website:	Website, Web, Mac, Macbook, Webpage, Windows, Fullscreen
Mobile:	Mobile, Phone, IOS, Iphone, Android, Application, Iphone_x
Tablet:	Tablet, Ipad, Ipadpro, Tablet responsive design
COLOR	
White:	White, Clean, Snow white, #ffffff, Light mode
Yellow:	Yellow, Golden, Orange, Gold
Red:	Red, Maroon
Pink:	Pink, Rose, Blush, Coral
Purple:	Purple, Violet, Magenta, Plum, Lavender
Blue:	Blue, DarkBlue, SkyBlue, Azure, Sky, LightBlue, Indigo
Green:	Green, DarkGreen, Aquamarine
Grey:	Grey, Silver, DarkGray, Dull, Gray
Brown:	Brown, Sandy, Chocolate, Sepia
Black:	Black, Dark, Black dark theme, Night mode
APP FUNCTIONALITY	
Music:	Music, Musicplayer, Apple_music, Playlist, Melody, Studio
Food & Drink:	Food, Restaurant, Drink, Salad, Recipe, Juice, Cola
Game:	Game, Videogame, Shooter, VR, Fighting game, Dota
Health & Fitness:	Fitness, Health, Healthy, Energy, yoga, Fitbit, Healthtracker
News:	News, Aid, Newspaper, Articles, Journal, Magazine
Sport:	Sport, Gym, Workout, Football, Trainer, Running, Exercise
E-commerce:	E-commerce, Store, OnlineShop, Ikea, Eshop, Webshop, Storages
Social Networking:	SocialNetwork, Blog, Messenger, Facebook, Instagram, Dating
Travel:	Travel, Trip, Tourism, Backpack, City guides, Flight, Car, Hotel
Weather:	WeatherApp, Temperature, Cold, Rain, Forecast, Umbrella
Lifestyle:	Fashion, Furniture, Real Estate
Education:	Education, E-learning, Maths, Languages, Course, Science
Reference:	Dictionary, Atlas, Encyclopedia, Documentation, Theme
Entertainment:	Movie, TV, Netflix, Youtube, Video, Art
Medical:	Medical, Healthcare, Drug, Blood pressure, Pharmacy, Patient
Books:	Books, DigitalReading, DigitalBookstore, Reading, Bookstore
Kids:	Kids, Children, Family, Cartoon, Boy, Teenager, Kids books, Toy
Finance:	Finance, Wallet, Bank, Business, Insurance, Currency, Coins
Utilities:	Calculator, Clock, Measurement, WebBrowser, Alarm, Calendar
Navigation:	DrivingAssistance, TopographicalMaps, PublicTransitMaps
SCREEN FUNCTIONALITY	
Landing Page:	Landingpage, Landing page, Welcome, Homepage
Login:	Login, Signin, Onboarding
Signup:	Signup, Registration, Account, Createaccount, Verification
Checkout:	Checkout, Payment, Credit card, Purchase, Cart
Search:	Search, Job search, Searchengine, Searchbar
Profile:	Profile, Illustration, Mockup, Portfolio
Contact Page:	Contact, Contactpage, Consultation, Faq, Help
SCREEN LAYOUT	
Dashboard:	Dashboard, Desktop, Control panel, Admin panel
Form:	Form, Input fields, Form elements
Table:	Table, Summary table, Pricing table
List:	List, Paylist, Dropdown, Tracklist, Shopping list
Grid:	Grid, Logogrid
Gallery:	Gallery, Drawingart, Photography, Exhibition
Toolbar:	Scroll, Toolbox, Tools, Menu, Navigation bar, Navbar
Chart:	Chart, Pie chart, Flow chart
Map:	Map, City map, Heat map

Table 2.2: The categorization of UI-related tags.

the UI design tagged with “e-commerce”, “e commerce design”.

We adopt a semi-automatic method [Chen et al., 2017b] to help normalize these morphological forms. To cover means of conceptual-semantic and lexical relations, we use a pre-trained word embedding NLTK model [Loper and Bird, 2002], which is a large lexical database grouped with nouns, verbs, adjectives and adverbs. We first encode its semantic by converting tag into vector. Then, we calculate a vector distance to measure the conceptual semantic similarity. For example, ("restaurant", "recipe"), ("shop", "store") are recognized. Next to further determine the lexical relations, we measure a similarity based on the string edit distance [Levenshtein, 1966] . For example, ("color", "colorful"), ("shop", "shopping") have a high similarity which are recognized as a cluster. Furthermore, to discriminate the abbreviations, we define a set of rules e.g., the character of the abbreviation must be in the same order as they appear in the full name (ui, **u**ser **i**nterface), the word without sign is the same as non-sign word (ecommerce, e**-**commerce), etc. Example of some most frequent morphological forms are listed in Table 2.3.

2.2.3 Tags prediction

Although our approach mentioned in last section can boost the performance of UI retrieval by normalizing the existing tags from designers, some UIs are still missing important tags which makes them totally unsearchable. Therefore, a method to recover the missing tags of existing UIs is necessary. However, due to the existing UI designs are too diverse, we could not find a code-based heuristic to identify tags from UI design images with high accuracy based on the labeled data. So, we propose a hybrid deep model modeling both the visual and textual information to predict missing tags of the UI design. Additionally, to understand hybrid our ensemble model make its decisions through the visual information, we apply a visualization technique (Saliency Maps) [Simonyan et al., 2013] for understanding which part of the figure leading to the final prediction (Figure 2.6).

2.2.3.1 Dataset preparing

We leverage the tag categorization during the creation of the UI semantic vocabulary as the label, and corresponding UI design images and attached tags as the training data. Although we can simply regard all tags as equal to train a multi-class classifier for tag prediction, we train a binary classifier for each tag label. This is because, each UI design may own many semantic tags such as "website", "black", "sport", "landing page" and "grid", and these tags are not exclusive from each other. Furthermore, binary classifiers benefit the system extensibility for new tags, as we only need to train a new binary classifier for the new tag without altering existing models for the existing tags. For training each binary classifier, we take the UI design with that tag as the positive data. And we take the UI design attached with same-category tags in Table 2.2 as the negative training data, as tags in different categories are not exclusive. To overcome the imbalance problem, we balance the number of positive

STANDARD(#)	ABBREVIATION & SYNONYMS
ui (61309):	*user interface, user_interface, user-interface design, uidesign*
website (28009):	*web, websites, webpage, website development, web design*
ux (24209):	*user experience, uxdesign, ux_design*
mobile (8554):	*mobiledesign, mobile phone, mobile_design, smartphone*
illustration (7159):	*illustation, digital_illustration, kids_illustration*
app (5887):	*apps, application, app development, app design*
landing page (5536):	*landing-page, landingpage, landing page design*
minimal (4938):	*minimalism, minimalistic, minimalist*
ios (4741):	*ios8, ios9, ios11, ios_design*
iphone (4736):	*iphone x, iphonex, iphone 7, iphone_8*
icon (4230):	*icons, icon design, icon pack*
logo (3704):	*logo design, logos, logotype*
food (2881):	*fastfood, food_blog, junk_food, doughnut*
clean (2723):	*clear, clean_design*
flat (2481):	*flat design, flat-design, flat-art*
interaction (2402):	*interactive, microinteraction, interaction design, user interaction*
dashboard (2141):	*dashboard design, dashboards*
branding (2071):	*branding design, rebranding, selfbranding*
sketch (2060):	*sketching, sketches, adobe_sketch*
ecommerce (1974):	*e-commerce, online commerce, shopping*
vector (1940):	*vectors, vector art*
product (1841):	*products, product page, product detail*
typography (1820):	*interface typography, 3d_typography*
gradient (1671):	*gradients, gradient design, blue gradient*
gif (1441):	*gifs, looping_gif*
layout (1400):	*layout design, layouts*
concept (1378):	*conceptual, concepts, concept art*
motion (1361):	*motion graphics, motion design*
responsive (1347):	*responsive design, response*
music (1251):	*music player, musician, musical, concert*
restaurant (1221):	*restaurants*
profile (1204):	*profiles, user_profile, userprofile*
travel (1197):	*travelling, travel agency, travel guide*
animation (1194):	*animations, 3danimation, 2danimation*
simple (1108):	*simply, simplistic, simple_design*
graphic (1047):	*graphics, graphic design, graphicdesigner*
color (1000):	*colors, colorful*
white (988):	*whitelabel, white design, white theme*
login (919):	*log_in, sign_in, login screen*
modern (915):	*modernistic, fashionable*

Table 2.3: The 40 most frequent UI related tags with their abbreviations and synonyms and in brackets indicate the number of appearance.

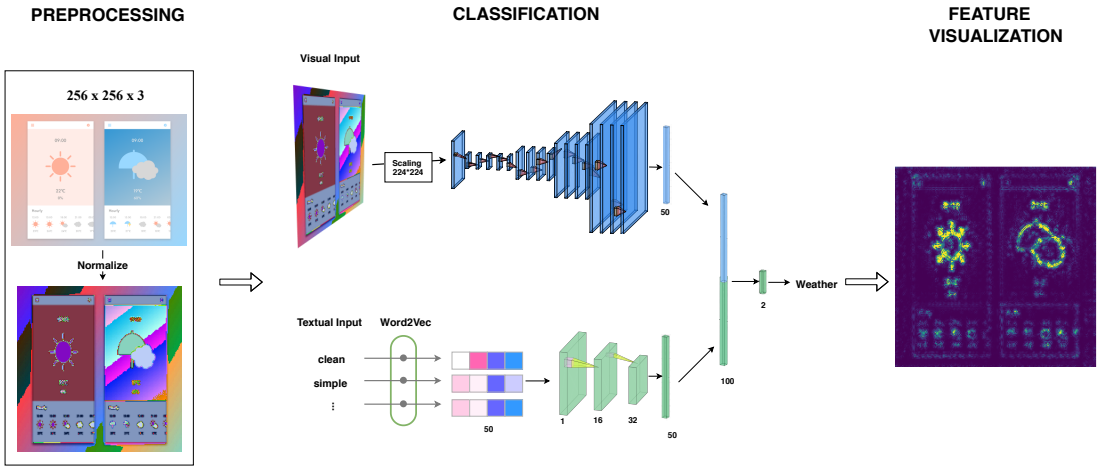


Figure 2.6: The architecture of our tag prediction model.

data and negative data in the training dataset.

Textual preprocessing For each tag, we merge similar tags based on the categorization of tags in Table 2.2 and the morphological form in Table 2.3 to decrease the ambiguity. For example, the tag "traveling", "travel guide", "trip" are all merged to tag "travel". We further apply feature selection method to accelerate training, i.e. removing unneeded, irrelevant and redundant attributes from data. We manually check 100 tags, and find that single character and simple number do not provide any meaningful information to the data. Hence, we regard these as unneeded and irrelevant attributes, which will be excluded on training data.

Visual preprocessing For each image, we normalize the pixel value [Finlayson et al., 1998] to improve neural network stability. The goal of normalization is to change the values of pixel value in the image to a common scale (i.e., 0 to 1), without distorting differences in the ranges of values. First, we sum up the pixel values in each channel (R, G, B). For each channel we compute the mean and standard deviation. Then, we divide mean and standard deviation by 255 (e.g., in our case mean is [0.4914, 0.4822, 0.4465] and standard deviation is [0.2023, 0.1994, 0.2010]). We apply normalization value on each image to accelerate training process.

2.2.3.2 Model Architecture

We propose an end-to-end deep fusion model to jointly learn visual and textual information. The architecture of the proposed network is shown in Figure 2.6, which contains two main components. One CNN model takes the GUI design image as the input for extracting the visual features. The other CNN model takes the existing tags as the input for extracting the textual information. We then merge both information sources to the final fully-connected layers for predicting the class of the GUI design.

Capturing Visual Information Convolutional Neural Network (CNN) [LeCun et al., 1998; Krizhevsky et al., 2012] is a class of deep neural networks. It occupies a

dominant position in analyzing visual imagery. A CNN is comprised of one or more convolutional layers with pooling layers and then followed by one fully-connected layer. The essence of CNN is the convolution of an image by using a kernel (K) (e.g., 3×3 matrix) and sliding around the image to capture all the features in the image. To control the overfitting, pooling layer is used because it summarises the features in the kernel, resulting in reducing the amount of parameters and computation in the network.

Different from the original CNN model, we adopt more advanced one, ResNet (Residual Neural Network) [He et al., 2016] with skip connections among the convolutional layers so that the model can be designed as very deep. But due to the limitation of our dataset and deep structure of the model, we cannot train the ResNet directly with our small-scale dataset. Therefore, we adopt the existing pretrained model which already distills the visual features from the ImageNet [Deng et al., 2009], the largest image dataset including more than 14 million natural images. Based on the pre-trained ResNet, we further use our dataset to fine-tune the ResNet to adjust the parameters in different layers.

Capturing Textual Information Apart from the GUI design picture, designers also attach some tags to index their picture. According to the observation in Figure 2.5, we find that some tags co-occur frequently, indicating some implicit relations among them. To predict the missing tags of a certain GUI design picture, we also take the existing tags into the consideration in our model. The classical approach of solving textual problems is one-hot encoding [Harris and Harris, 2010]. However, this approach has multiple drawbacks. First, the dimensions of one-hot encoding is linear to the unique words in the dataset. If the dataset has 1000 unique words, then one-hot-encoded vector representation will have 1000 dimensions. Second, one-hot-encoded vector representation is mostly empty (zeros). This is because, designers can use maximum 20 tags to describe their UIs, thus the vector representation will have maximum 20 1's and other 0's. Suppose we have 200 unique words, then the ratio between 1 and 0 is 1:9. The unbalanced representation is inefficient in computation and hard to converge. To solve this problem, we apply word embedding model [Mikolov et al., 2013a]. Given a set of existing tags, we first adopt the pre-trained word embedding model[Pennington et al., 2014] to convert each tag into a vector in terms of encoding its semantic. Then we synthesize all tags' vector representation into a 50×50 map (see in Figure 2.6) and feed it into a 2-layer CNN for extracting the salient textual information. Note that instead of using the trained word embedding, we use pre-trained one. This is because, our word embedding model is trained based on our collected data, however, there are numerous new words that are not contained in our data. In order to cover as much information as possible, we use pre-trained word embedding model that is trained on large database, such as Wikipedia 2014[wik, 2019] and Gigaword 5[Parker et al., 2011]. The experiment also proves our concept in Section 2.3.2. Apart from the word embedding, we also use Convolutional Neural Network to train tag information rather than the widely-used Recurrent Neural Network (RNN) [Mikolov et al., 2010]. This is because RNN takes account in the order (e.g., the result of $< a, black, cat >$ and $< cat, a, black >$ is

significantly different). As there is no explicit order among tags, RNN is not appropriate. In contrast, order is not a critical factor in CNN, as the kernel K extracts all the features. In addition, the max pooling in CNN overcomes the problem of zeros padding.

Combining Visual and Textual Information for Prediction Based on the image and tag input to the ResNet and text CNN, we obtain the embedding vectors which encodes the visual and textual information. To determine the intensity of information, we conduct an experiment on different embedding vectors. As a result, we find that they capture equally amount of information. Base on the empirical study in Section 2.3.1, we set the embedding vectors to 50 respectfully. We then concatenate both vectors and feed them into another fully-connected layers for the final prediction. Therefore, we link both the ResNet, text CNN and final fully-connected layers as an end-to-end system for taking the UI design pictures and attached tags as the input, and output the predicted missing tags for them.

2.2.3.3 Prediction Visualization

One of the main drawbacks of a deep learning system is the uninterpretability i.e., how the model gives a certain prediction. To gain insight into our classifier for the prediction results, we adopt a visualization technique [Simonyan et al., 2013] which calculates a saliency map of the specific class, highlighting the conclusive features captured by ResNet. Given the prediction result, the activities of decisive features in intermediate layers are mapped back to the input pixel space by class saliency extraction [Simonyan et al., 2013]. For one input UI design image, the final encoding of ResNet is mostly based on the weight of each pixel. Given the prediction result, the derivative weight map can be computed by back-propagation. Then the elements of weight are rearranged to form the saliency map. As shown in the final part of Figure 2.6, it can be seen that the ResNet model predicts the tag "Weather" for that input design due to the existence of the sun, umbrella and cloud in the UI design image.

2.3 Results

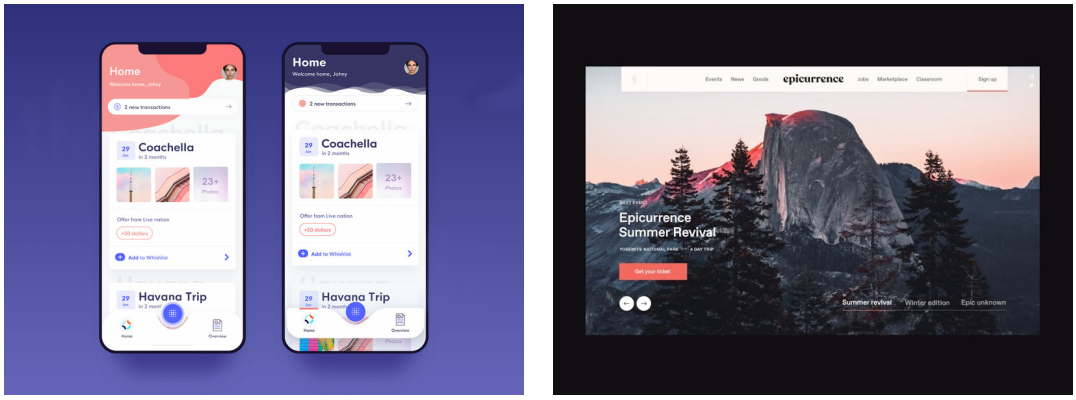
In this section, we will first analyse the performance of our model, as well as discussing our finding. Then, we compare the derivations of our model with several basic baselines including deep-learning and machine-learning. Finally, we express the model behaviours on some example UIs to see how it performs on the real world data.

2.3.1 Model Performance

Given all the data assigned to each tag, we take 90% of them as the training data, and the other 10% as the testing data. As we train a binary classifier for each tag, we adopt the accuracy as the evaluation metric for the model.

Image vectors	Tag vectors	Accuracy
10	90	0.5972
20	80	0.7056
30	70	0.6528
40	60	0.8134
50	50	**0.8272**
60	40	0.8222
70	30	0.8194
80	20	0.7981
90	10	0.7715
32	32	0.8008
64	64	0.8130

Table 2.4: A result of classification accuracy with different number of embedding vectors in Image and Tag



(a) Mobile design (b) Website design

Figure 2.7: A illustration design example of "mobile" and "website"

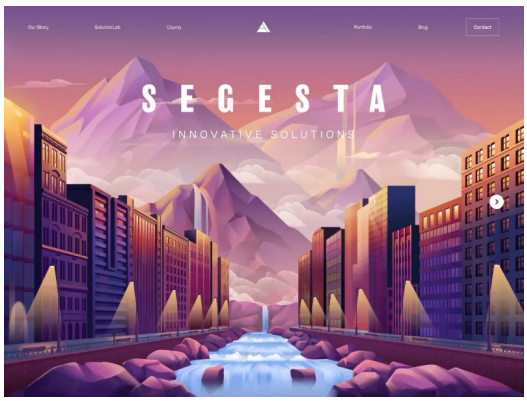
As discussed in Section 2.2.3.2, the intensity of visual and textual information is a critical thing to determine. We test different number of embedding vectors for Image and Tag. From Table 2.4, we can see that image and tag capture equally amount of information. Thus, half and half is an ideal distribution. Furthermore, we set 50 as each embedding vectors as 50-50 achieves best performance among experiments.

A detailed results of our model can be seen in Table 2.5. Our model (Tag+Image) can achieve the average accuracy as 0.8272. We can see that it performs well on "Platform" topic, which model can recover "mobile" and "website" in 0.9402 and 0.9171 accuracy, respectively. This can be explained in several reasons. First, the size of the dataset in "mobile" and "website" is larger than others (i.e., 26,466 and 22,130). Large volume dataset provides diversity of instances. Therefore, model is able to learn a wider variety of strategies to recognize the patterns, hence, improving the generalization of model. Second, the big data is critical for deep learning model. In general,

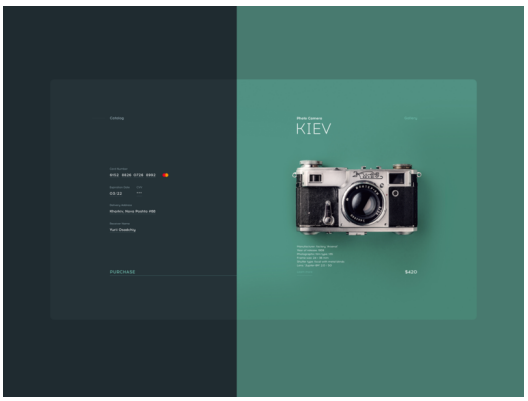
CLASSES	ACCURACY						
	Histo +SVM	**Histo +DT**	**ResNet -pretraind**	**ResNet +pretraind**	**Tag only**	**Tag(trained) +Image**	**Tag +Image**
App Function							
music	0.6636	0.5545	0.6727	0.7909	0.8545	0.8589	**0.8909**
food&drink	0.5765	0.6294	0.7529	0.7882	0.7706	**0.8294**	**0.8294**
ecommerce	0.5565	0.5726	0.6895	0.7460	0.8306	0.8349	**0.8710**
finance	0.5655	0.5833	0.6964	0.7500	0.8274	0.8452	**0.8512**
travel	0.5211	0.5842	0.7316	0.7053	0.8053	**0.8915**	0.8474
game	0.5814	0.5814	0.8062	0.7984	0.7597	**0.9000**	0.8605
weather	0.5745	0.7021	0.7447	0.7872	0.8085	**0.8842**	0.8298
sport	0.4220	0.6147	0.6147	0.6239	0.7064	**0.8723**	0.7798
Color							
yellow	0.5865	0.7596	0.7404	0.7404	0.6442	**0.8000**	0.7500
red	0.6667	0.7083	0.8194	**0.8472**	0.6111	0.7778	**0.8472**
pink	0.7609	0.6522	0.7826	0.7391	0.6522	**0.8496**	0.8261
blue	0.6600	0.6800	0.7700	0.7400	0.6800	0.7857	**0.8700**
green	0.7000	**0.8714**	0.8286	0.7714	0.6571	0.8043	0.7857
white	0.6111	0.6111	0.7778	0.7333	0.7333	**0.8778**	0.7888
black	0.6241	0.6015	0.8496	0.8271	0.6617	0.7981	**0.8571**
Screen Function							
landing page	0.5465	0.5346	0.7106	0.7017	0.7947	0.7727	**0.8115**
signup	0.4907	0.5556	0.7731	0.7130	0.7361	**0.8496**	0.7778
checkout	0.5545	0.4182	0.6000	0.7091	0.7545	0.7487	**0.8000**
profile	0.4667	0.5538	0.5487	0.6513	**0.9026**	0.8102	0.7590
Screen Layout							
dashboard	0.5867	0.6067	0.7600	0.7933	0.7867	0.8636	**0.8800**
chart	0.6061	0.6667	0.7121	0.7424	0.8485	**0.8867**	0.8030
form	0.5429	0.5000	0.6857	0.7429	0.5714	**0.7857**	0.7714
list	0.6136	0.5909	0.7045	**0.9091**	0.6364	0.7568	0.8182
grid	0.5000	0.5811	0.6351	0.6486	0.7162	**0.9091**	0.7432
Platform							
mobile	0.7785	0.7847	0.8356	0.7954	0.9250	0.9398	**0.9402**
website	0.7513	0.7481	0.8418	0.8224	0.8837	0.9012	**0.9171**
Average	0.5965	0.6249	0.7342	0.7545	0.7522	**0.8468**	0.8272

Table 2.5: Tag classification accuracy for five categories in different methods.

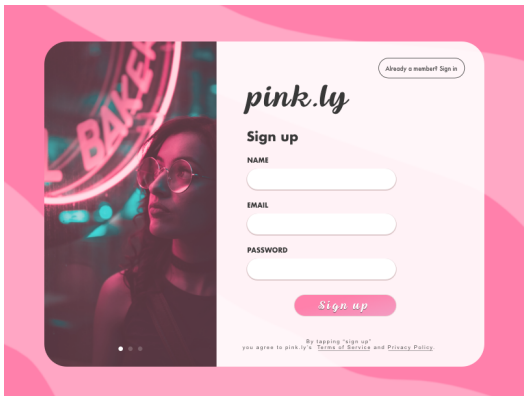
deep model has thousands and sometimes even millions of parameters. Therefore, if the dataset size is smaller than the parameters, the model can simply memorize the patterns of instances instead of learning. Thus, it misleads accuracy detracts from the predictive models we build. Third, the simple pattern can improve the accuracy. For example, Figure 2.7(a) shows an example of mobile and website artwork. We can see that mobile has a small boundary (i.e., Iphone layout), on the contrary, website has a large and explicit boundary (i.e., a contrast between background and design). Therefore, a simple pattern allows the model easily recognize its label.



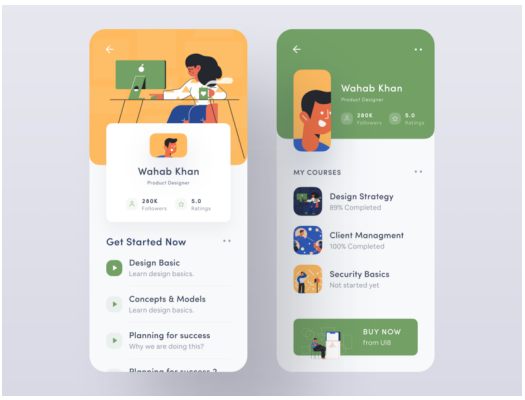
(a) Landing page



(b) Checkout



(c) Signup



(d) Profile

Figure 2.8: Examples of the three kinds of prediction errors

In contrast, we can see that the model does not perform well on "Screen Function", which the average accuracy is 78.71%. This is mainly because of three reasons. First, the pattern is not distinct. There is no clear features for our model to predict it correctly. For example, "Landing Page" design in Figure 2.8(a), it looks more like an art illustration. Therefore, the model is hard to classify that the design as "Landing page" or not. Second, the pattern is not prominent. The example design of "Checkout" (Figure 2.8(b)) and "Signup" (Figure 2.8(c)) shows that the feature is the word "PURCHASE" and "Sign up" respectively. However, the feature is relatively small

to the whole design. Therefore, it is highly possible that the feature "disappears" after several convolution layers. In other words, the model is hard to learn small features. Third, many UIs are correlated to indicate a tag. Rather than isolated to each other, there is a connection between many UIs. For example "Profile" design in Figure 2.8(d), while single UI can not be classified as "Profile", the whole design can be regarded as "Profile". However, the model does not have the ability to recognize the pattern unless it can learn the connection.

2.3.2 Baseline Comparison

From Table 2.5 we can speculate that our result outperforms the baselines except Tag(trained)+Image. Next, we split the baselines into two parts (i.e., deep-learning and machine-learning). Note that the training and testing configurations for these baselines are the same with that of our model.

deep-learning ResNet has a huge number of parameters (i.e., 11.7M parameters in ResNet-18). Since the size of our dataset is smaller than the number of parameters in ResNet, it greatly affects the ability of model to generalize, often results in overfitting. Fine-tune [Yosinski et al., 2014] on pre-trained model (i.e., train the last few layers) is a widely used technique to solve the problem. As we observe Table 2.5, the usage of the pretrained ResNet based on the ImageNet dataset increases the accuracy from 0.7342 to 0.7545. Thus, the fine-tune process improves our model. From Table 2.5, we can also see that the combination of the visual information and textual information can lead to better prediction accuracy than using any single information.

Based on our UI tags in Section 2.2.1, we adopt Common Bag of Words model (CBOW) [Mikolov et al., 2013b] to train a specific UI word embedding. And we apply the trained word embedding to improve our model. We speculate that Tag(trained)+ Image outperforms than our model. Although trained word embedding achieves better performance, we still use pre-trained (Glove [Pennington et al., 2014]). This is because, as discussed in Section 2.2.3.2, the diversity of our collection is limited, so that the trained word embedding gives a false sense of a highly accurate model. To prove our concept, we test the Tag(trained)+Image model on new crawled image. We manually check the accuracy and the predicted result. We found that the accuracy tends to close to accuracy of ResNet model. In other words, the Tag(trained)+Image model loses the ability of capturing textual information. Therefore, in order to improve the generalization of model, we try to cover as many words as possible. In conclusion, we adopt Tag+Image model using pre-trained word embedding as our tag prediction model.

machine-learning We set up several basic machine-learning baselines including the feature extraction with two machine-learning classifiers. Color histogram (Histo) [Wang et al., 2010b] is a well known feature representation in image retrieval. Color histogram for an image is extracted by quantizing the colors within the image and counting the number of pixels of each RGB (red, green, blue) channel. To estimate how well our deep neural network preforms, we adopt the widely used classifiers: support vector machine (SVM) [Cortes and Vapnik, 1995] and decision

tree (DT) [Quinlan, 1983]. These two classifiers dominate binary classification problem. The comparison result is shown in Table 2.5. The traditional machine learning method based on the human-crafted features can only achieve about 0.6 average accuracy, while our model can achieve the average accuracy as 0.8272 i.e., the 32.4% boost than the best machine-learning baseline. Thus, the conventional machine learning based methods are not capable to deal with the complexity of the UI design images.

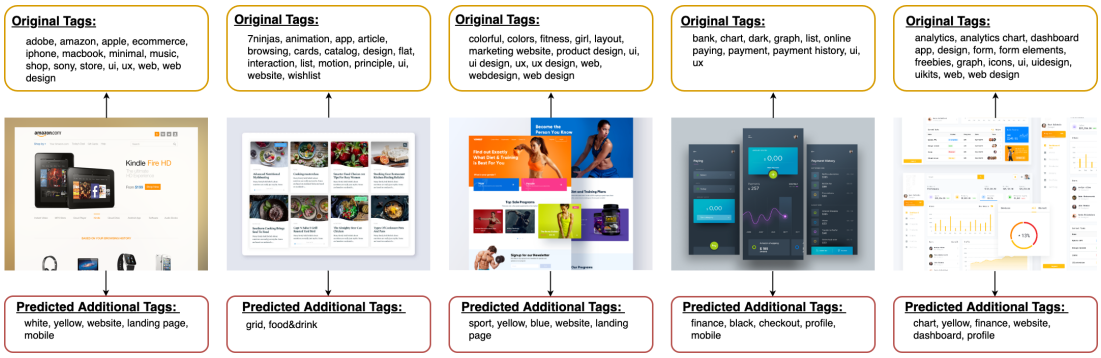


Figure 2.9: The predicted tags by our model for complementing the original tags.

2.3.3 Model Behaviours

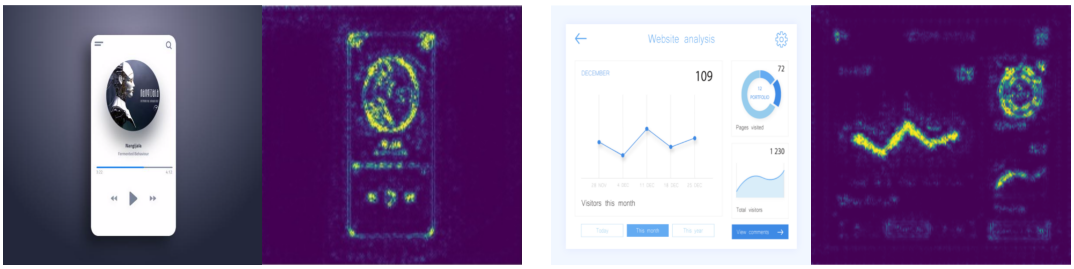
We show some missing tags recovering examples in Figure 2.9. For example, for the PLATFORM category, the model can predict "mobile" correctly in the fourth example. For the COLOR category, "white" is predicted in the first example and "black" is predicted in the fourth example. For the APP FUNCTIONALITY category, the model can predict "food&drink" in the second example and "sport" in the third example. For the SCREEN FUNCTIONALITY, it can recover tags like "landing page" in the first and third example, and "checkout" in the fourth example. Lastly, for the SCREEN LAYOUT, "grid" is recovered in the second example and "dashboard" in the fifth example. Note that the predicted additional tags are generated by semantic in Section 2.2.2 and model in Section 2.2.3. Apart from the benefit of recovering tags, we can take advantages of the model in the real application/website like Dribbble to do a recommendation. For example, as our model can predict tags, we can recommend potential tags to users before they upload their designs. Thus, it can not only help users locate tags, but also proactively ensuring the quality of the design labels.

To show how our deep learning model works well for predicting the tag for the given UI design, we visualize the features learned in our model. We traced the input through the computational graph in order to find out feature performance in all the layers of our models (as shown in Figure 2.10). We can notice that they, in some way, makes sense; for example, in the first layer few basic feature is displayed (i.e., bright edges). The deeper the layer, the clearer the feature. At the last layer, we can see that the feature is explicit which is a game controller, resulting in predicting a "game" tag. To further gain insight on more tags features, we show several examples

(a) origin (b) layer-6 (c) layer-9 (d) layer-15 (e) layer-18

Figure 2.10: Visualization of the salient features in different layers.

(a) food&drink (b) travel

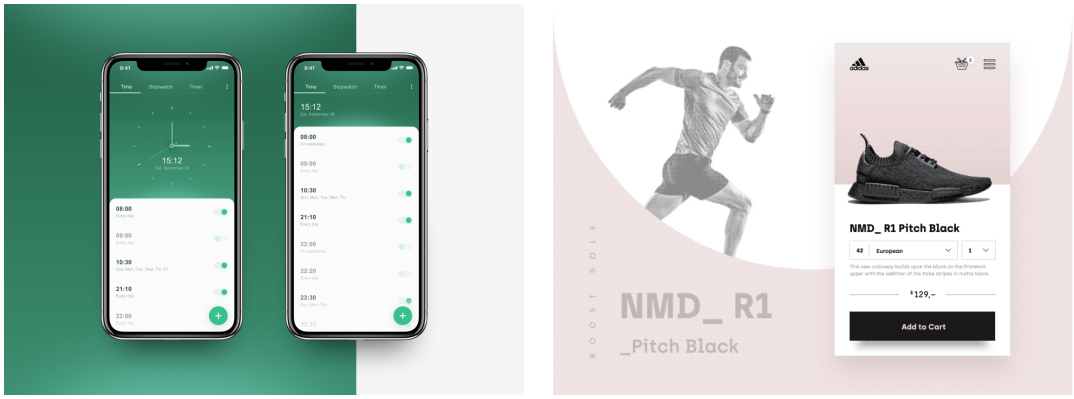


(c) music (d) chart

Figure 2.11: Visualization of the salient features in our model leading to the final predictions.

in Figure 2.11. For example, (a) the burger is highlighted in the UI to predicted tag “food & drink” (b) our model has paid much attention to the mountain to predicted tag “travel” (c) our model spots the play button (i.e., widely appears in the music player apps) to predicted tag “music” (d) our model emphasizes the chart line to predicted tag “chart”.

To identify the common causes of prediction errors, we further manually check the wrong predictions in the dataset. As observation, there are five reasons leading to the wrong prediction. The top three reasons are described in Section 2.3.1; indistinct pattern, not prominent pattern and implicit connection. And we further describe another two reasons. First, the negative data is not really negative. We suppose that there is no same-category tags (i.e., if the design is tagged with one tag, then it is not tagged with another tag in the same category). For example, in APP FUNCTIONALITY category, we have "music", "food&drink", etc. If the design



(a) Same-category tags (b) Ambiguity in the tag vocabulary

Figure 2.12: Examples of the another two kinds of prediction errors

is tagged with "music", then it does not contain "food&drink". It works in most case, however, it still have some different cases, resulting in "noisy" data. For example, in Figure 2.12(a), it is hard to classify the design in any color tag, as the main color can be "green" and "white". Second, there is ambiguity in the tag vocabulary since a tag may have multiple meanings. For example, the tag “sport” is usually associated with fitness applications, but it can also indicate sports wears. As in Figure 2.12(b), the design is clearly a e-commerce app, but it is tagged with “sport” without any “e-commerce” related tags.

2.4 Gallery system design and construction

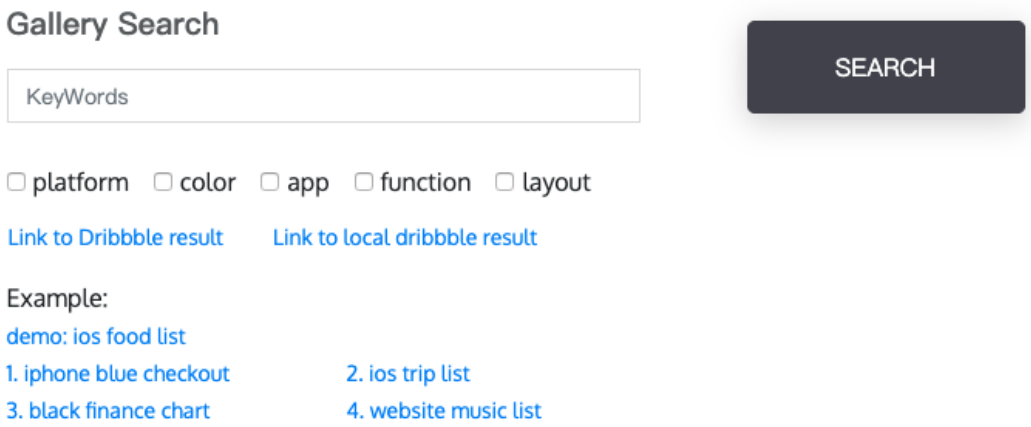


Figure 2.13: A design gallery system including search filter and baselines comparison.

With the graphical design collected in Section 2.2.1 and the associated tags, we are now ready to build a novel design gallery system that can satisfy the designers' need for advanced design search and knowledge discovery. We implement a proof-of-concept prototype of our GUI design gallery in the web application http://mui-collection.herokuapp.com/test. An illustration of our gallery system is shown in Figure 2.13. In the following subsections, we first introduce our gallery database, and then describe the advanced search techniques such as search filter, existing platforms comparison and ranking system in detail.

2.4.1 Gallery Database

We randomly crawl another 20,037 designs from Dribbble in 20th Sep, 2019. We filter out the non UI designs as discussed in Section 2.2.1. As a result, we obtain 16,546 UI designs as our gallery database. Note that we use neither training nor testing dataset in Section 2.2.1. This is because, the new crawled designs are fresh so that we can estimate how well the model performs on the new inputs (see the reason in Section 2.3.2). Moreover, design style evolves and constantly changes over time (e.g., a popular UI design in the 2000's is now outdated), as well as tags. Since the design and meta data are crawled recently, we can estimate the generalisation of our model given that it is facing the latest design and tags.

2.4.2 Search Filter

We provide a query searching box that enables users to specify criteria about an design of interest. For example, if user wants to see the designs of "Sport", he can enter keyword in the searching box to retrieve designs which are related to topic "Sport". The retrieved result depends on both user pre-defined tags and predicted tags.

Additionally, in order to refine and decompose user requirements, we apply a categorical filter based on Table 2.3. For example, if the user selects "Color" filter, the retrieved results will be split into "white", "yellow", "red", etc. Therefore, user can not only browse to search specific designs, but also customize search results to find exactly what they want. In addition, it helps users to understand the design demographics. We will further elaborate the usefulness of this filter in Section 2.5.3.

2.4.3 Existing Platforms Comparison

To efficiently evaluate our results, we compare with two baselines. One is the results in Dribbble with the same query. However, the benchmark is unequal. This is because, the database in Dribbble is several dozens of times larger than us (16,546 UI designs). And the database keeps increasing day by day. Thus, we provide another baseline. We simulate Dribbble search method (i.e., a fully matched search on tag, color, title, etc.) on our database. Therefore, under relatively equal conditions, we can estimate the performance of our model precisely.

2.4.4 Ranking System

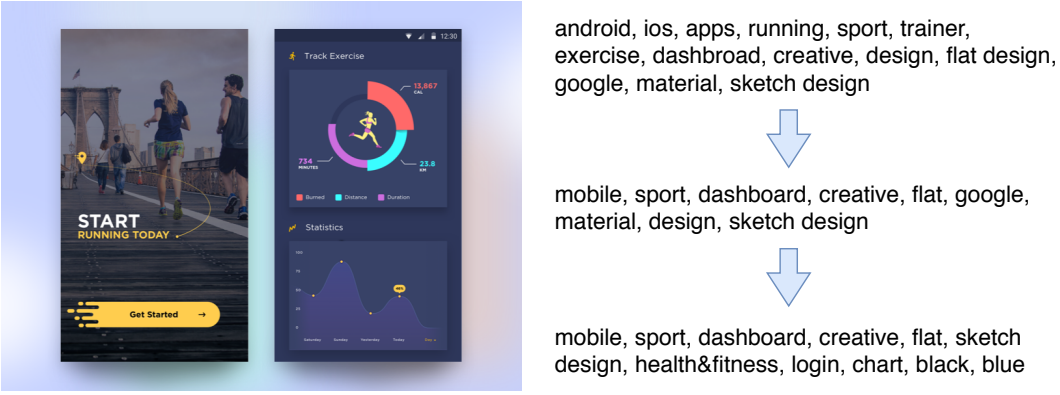
We find that some designers upload their designs with irrelevant tags in order to gain attention. For example, assume that "Olympic" is the most popular tag currently. Designer may upload all of his designs with this tag, so that his designs will be easier to search. In order to prevent irrelevant tags affect user experience, we take advantage of model prediction in Section 2.2.3. First, we feed all the gallery designs in Section 2.4.1 into the model and predict the probability of each tag in Table 2.2. We adopt a ranking system based on the probability, in terms of degree of confidence (i.e., how confidence the prediction is). Therefore, the most relevant designs will be appeared at the top of the gallery and the least relevant will be at bottom. Note that although our model predict the predefined tag in a low degree of confidence, we still assume that the tag relates to the designs. This is due to two reasons. First, even though our model can predict most of the tags correctly, the model still makes mistakes as discussed in Section 2.3.3. Therefore, there is a chance that our model predicts incorrectly. Second, removing results may cause losing information. As a matter of fact, designs at bottom have little impact on users because users rarely turn to the bottom. Therefore, in order to balance the quality and quantity, we decide to keep all the designs. For example, assume a designer tagged his design as "Sport" and our model gives this tag a low degree of confidence. We will still return this design in topic "Sport", but appears at bottom.

2.5 Informal Feedback from Designers

To understand how the design gallery would be helpful in the real world environment, we conducted an informal walkthrough with 10 Master and final-year Bachelor students and 4 professional UI designers, including one interaction designers and one visual designer from Google, and the other two visual designers from Huawei. These designers acknowledge that they regularly (at least once weekly) visit design sharing websites like Dribbble. They appreciate the creativity and aesthetics of high-quality UI design artworks on Dribbble, which inspire their own design work. To collect informal feedback on the idea of predictions from our tag recover model (Section 2.2.3.2), we introduced our gallery to the users, including functions and general purpose. We further asked them to envision new ways they could use our gallery in the real design context beyond the basic functionality. In sum, our design gallery was well received by users. They was impressed on the performance of tag recover model and also provided us several incited comments to improve our gallery.

2.5.1 Visualizing Vocabulary

As the contributors of Dribbble site, some designers do not pay much attention to their tagging, though it is crucial for the GUI index and searching. One designer from Huawei mentioned us that he did not intentionally use the semantic words (e.g., functionality, layout) to annotate his design works, as he did not know much



(a) a sample design (b) tags improvement process

Figure 2.14: A process of annotating design works

what tags are there and which tags to use. Instead, he attached his UI design with some fancy tags like "2019trending" to make his design more attractive to other designers. He was impressed on our vocabulary that is in the full compliance with his requirements and described a particular use case in which our structural vocabulary can be helpful of annotating his design as shown in Figure 2.14. In his work of designing a sport app, he attempts to upload his design in Dribbble to gain feedbacks from other professional designers. First, he preferred annotating few possible tags as usual. Then, with the help of vocabulary, he tried to standardize his tags which is to remove the related tags (i.e., removing running, trainer, exercise as they are correlated with sport). Meanwhile, he checked if any abbreviations, synonyms and misspellings in his tags (i.e., dashbroad). Finally, he browsed our vocabulary table to get some inspirations on the missing tags (i.e., health&fitness, chart, login), as well as removing any tags lacking clarity of meaning (i.e., google). Our structured vocabulary of tags in Section 2.2.2 assists him efficiently annotating his design works for other designers' retrieval.

2.5.2 Exploring Variations

The designers also pointed out the potential of the gallery for helping them systematically explore variations. They mentioned that there is a gap between existing UI tags and the query due to two reasons. On the one hand, some UIs are missing important semantic tags. For example, one Google designer was designing a website about Yoga. When she searched "yoga", there are many different kinds of returning results from Dribbble including the icons, charts, illustration which are not related to her designed UI or website. She had to manually filter out irrelevant ones, and that process took much time. But when she zoomed into a more specific query like "yoga website", only very few results were returned by Dribbble. On the other hand, designers may adopt different words to tag the same GUI, resulting in the difficulty of retrieval. Another designer in Google mentioned that he would use different words

Query	Number of Dribbble	Number of our gallery
ios trip list	14	**187**
iphone blue checkout	9	**39**
black finance chart	10	**50**
website music list	3	**16**

Table 2.6: A comparison on the number of retrieved images between our gallery and Dribbble [Dribbble, 2010]

in the same meaning for searching such as both “travel“ and “trip“ for related GUI, but that process is inefficient as many results from two queries overlap, and he may miss some potential synonyms.

They praised our gallery compared to Dribbble, our gallery is able to retrieve more relevant designs. As one Google designer was currently working on a trip application for IOS frame layout, in addition to make his design consistent with other UIs in the application, he requires a list view UI design. However, He found only a small number of designs in Dribbble (i.e., 14 designs) which is too little to find inspirations. In contrast, he applied the same query "ios trip list" in our gallery. Surprisingly, our gallery returns 187 designs. Some image retrieval results is shown in Table 2.6. Apart from the size of the retrieved results, he also noticed that our organization of the result is better than Dribbble. He was quickly got inspired by the top 20 designs in our gallery due to the ranking algorithm (see in Section 2.4.4) in our gallery. Thus, our gallery assists designers on not only adding more tags to the GUI design, but also organizing existing tags.

	Number of designs	Proportion
Grey	1	0.51%
Brown	2	1.01%
Pink	2	1.01%
Black	42	21.21%
White	48	24.24%
Green	10	5.05%
Blue	80	40.40%
Red	5	2.53%
Yellow	8	4.04%

Table 2.7: Result of query "social" applied with color split

2.5.3 Design Demographic

Many students report that the gallery can help them understand the design trends through the retrieved results. Students provides an example of how to use our gallery to help them design a social applications. Although these students have a systematic learning on designing user interface, but most of them did not have professional

experience on how to design a real life application. To fit the user preferences, they were struggled on the UI design style, such as the dominant color. However, they found that the main-stream design style in the market could benefit from the categorical split. They entered the query "social" applied with a color split. Our gallery returns 80 designs for blue and 48 designs for white as shown in Table 2.7. With the help of the demographic, they understood the dominant color for social applications are blue and white. The designers and students thought that our gallery could be useful in its current state even for helping them understand the relative design trends in real life.

2.5.4 Areas for Improvement

Several designers wondered whether the database could be extended/real-time. They found that although our gallery can return a large amount of high quality designs in the demo and example queries, it does not outperform in other queries. Although our gallery displays a large scale of static designs, it lacks of dynamic UI designs. Dribbble has a huge database of dynamic UI designs, including gif and video. Designers confirms the usefulness of dynamic designs in their work. It is a gold mine of UI designs. And the resource allows us to extend and strength our gallery.

Chapter 3

Component-Oriented Search

Online communities like Dribbble [Dribbble, 2010] and GraphicBurger [GraphicBurger, 2013] 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 chapter presents design gallery (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, design gallery supports novel ways for designers to collect, analyze, search, summarize and compare GUI designs on a massive scale. We quantitatively evaluate the quality of design gallery and demonstrate that design gallery offers additional support for design sharing and knowledge discovery beyond existing platforms. This chapter is structured as follows:

1. **Section 3.1** investigates the related work including searching techniques and component wirification.
2. **Section 3.2** proposes a workflow of our system including data collection, data pre-processing, component wirification and component post-processing.
3. **Section 3.3** evaluates our proposed model (Faster RCNN). We first introduce an evaluation metric, then evaluate the performance of model within screenshots.
4. **Section 3.4** introduces the gallery system design and construction, including multi-Faceted search, design demongraphics, design comparison and design sharing.
5. **Section 3.5** understands the usefulness of our model and gallery in a real design context via an informal interview with professional designers. We further discuss the areas for improvement.

6. **Section 3.6** discusses the implication of our system in three aspects: designers, design sharing platforms and the research community.

3.1 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 [Pinterest, 2018]. 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 [Inspired-ui, 2018], Pttrns [Pttrns, 2018]. 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 [Axure, 2018]. 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 [Uplab, 2018], 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 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 [Deka et al., 2017] uses reverse-engineering and crowdsourcing to collect a large dataset of app runtime GUI screenshots. Based on the static analysis, StoryDroid [Chen et al., 2019] can more efficiently collect the app GUI screenshots than Rico. Webzeitgeist [Kumar et al., 2013] 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 screenshots as the main design resources and collect runtime GUI screenshots to train an object detection model to extract GUI components from app screenshots. To the best of our knowledge, our work is the first to build a large-scale GUI component design gallery using app 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. [Chen et al., 2018b]. 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 [Ren et al., 2015]. A similar work to ours is the mobile UI reverse-engineering work by Nguyn et al. [Nguyen and Csallner, 2015]. 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. [Moran et al., 2018] 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 [Jo and Jung, 2016] 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. [Meier et al., 2014] in which the authors find the relationship between location search patterns and user requirements and study by Swearngin et al [Swearngin et al., 2018] 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 [Reiss et al., 2018; Behrang et al., 2018; Zheng et al., 2019]. Reiss [Reiss et al., 2018] parses developers' sketch into structured queries to search related UIs of Java-based desktop software in the database. GUIfetch [Behrang et al., 2018] customizes Reiss's method [Reiss et al., 2018] into Android app UI search by considering the transitions between UIs. Similar to Reiss's work [Reiss et al., 2018], Zheng et al [Zheng et al., 2019] 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.

3.2 Design and Implementation

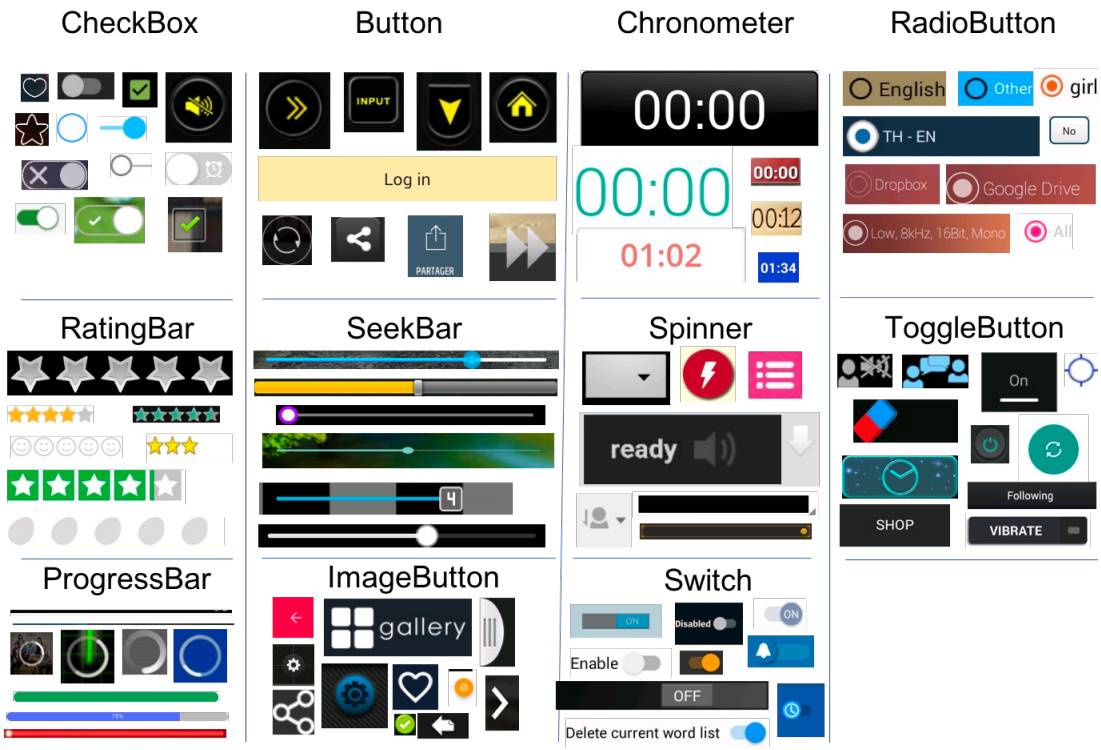


Figure 3.1: Examples of 11 types of UI components wirified in app introduction screenshots by our approach.

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). Figure 3.1 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. The goal of this work is primarily to collect as diverse elements from existing apps as possible. To obtain the GUI elements from the screenshots, we need a

method to decompose (i.e., wirify) app GUI screenshots into GUI components in the screenshots. It involves five procedures. First, we collect dataset from Google Play. Then, we apply data pre-processing technique on the basis of data like removing the duplicates to help manage the data. Good dataset accelerates the model training and also improves the results. Next, we train the model to wirify screenshots. We further apply post-precessing techniques on GUI components such as remove duplicate UI components, detect primary color etc. These techniques help users gain better insight on the usefulness of our model. Finally, in order to evaluate our model, we develop a website Design Gallery to demonstrate our retrieved results. We will further describe each procedures and methods in detail.

3.2.1 Data Collection

Google Play is an online Android Market which allows users to browse and download applications developed with the Android software development kit (SDK).

Application Metadata Collection For each application, it is attached with the corresponding meta data including a title, a review status, a rating, a downloads status etc. In order to collect application SDK and associated metadata from Google Play, we build a web crawler as discussed in Section 2.2.1.

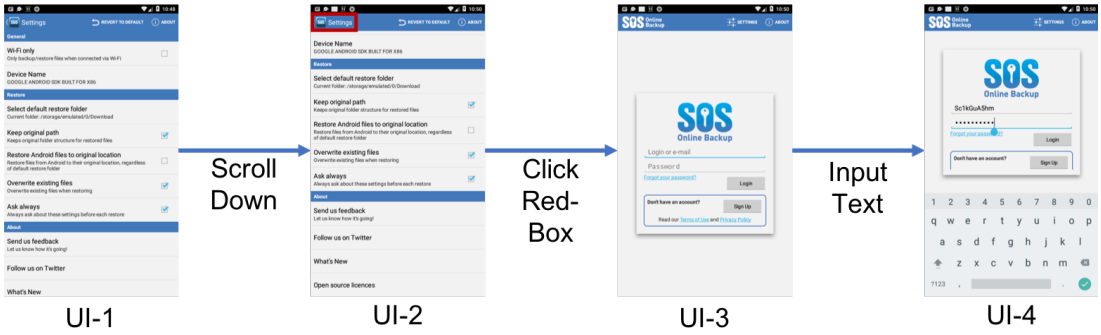


Figure 3.2: An illustrative example of simulating user actions on mobile app GUI proceeding from state UI-1 to state UI-4.

Screenshot Metadata Collection We use a data collection tool which is inspired by automated UI testing techniques [Su et al., 2017], to collect UI screenshots. The tool tries to identify executable GUI components (e.g., scrollable, clickable, editable) on the current UI and infer actions from the type of these components. It then emits the corresponding UI events to simulate user actions, resulting in automatically explore different parts of mobile app. The techniques can also simulate the interaction with the hardware buttons (e.g., back). As illustrated in Fig. 3.2, 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.

During such automated GUI exploration, the data collection tool uses Android UI

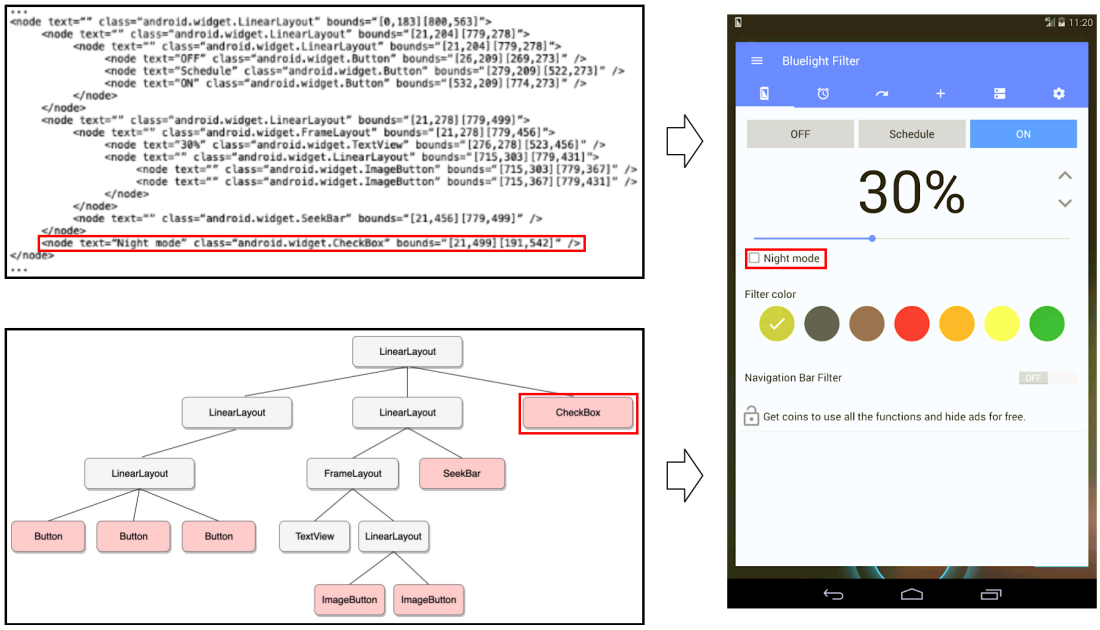
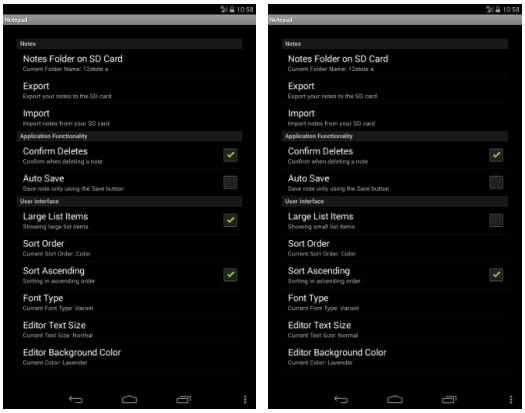


Figure 3.3: An illustrative example of the run-time GUI hierarchy and corresponding GUI screenshot

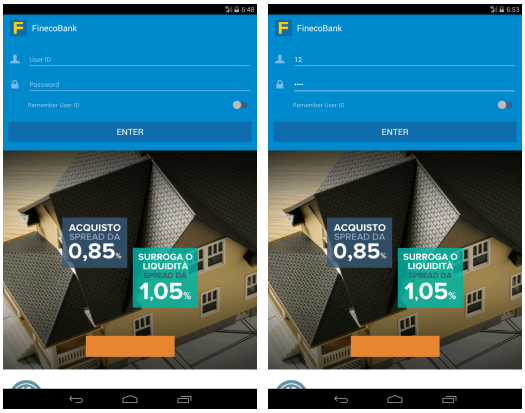
Automator [web, 2018] to automatically dump pairs of runtime GUI screenshots and corresponding runtime GUI component hierarchies. The runtime GUI component hierarchies are stored in XML files. Fig 3.3 shows an example about the UI and corresponding code (in the upper part), and the button outlined in 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.

3.2.2 Data pre-processing

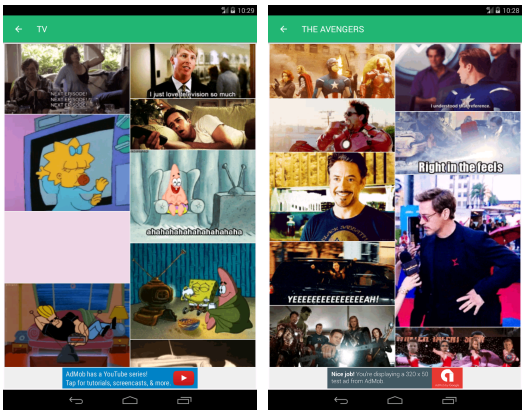
As discussed in Section 3.2.1, totally we get 68,702 GUI screenshots of 5,043 Android applications and their meta data (i.e., application and screenshot). However, as observation, we obtain great amount of similar/same screenshots (see in Figure 3.4). There are many reasons cause this. First, applications have design consistency. In Figure 3.4(a), we can see that designers apply same component in same position across different stages to obtain consistency. Second, applications have policy and privacy. As shown in Figure 3.4(b), we can see that we need an account to login to the application. Therefore, we get stuck in the login page and keep taking screen-



(a) design consistency



(b) app policy and privacy



(c) similar structure

Figure 3.4: An illustrative example of duplicate screenshots in the dataset.

shots on the same page.

When the data have a great amount of same instances, it is easy to get overfit. For example, assume you have two same screenshots, A and B. After random shuffling and splitting, A is divided into training data and B is divided into testing data. Thus, as long as the model can "recognize" A, it can "recognize" B. In other words, model can simply memorize instance, not learn. Thus, the model will get very high accuracy on in-sample testing but out of sample testing will be much lesser. Therefore, removing duplication is essential. We remove the duplicates by calculating the similarity between screenshots. Since word processing is faster than images, we calculate the similarity on XML structure to obtain efficiency. We firstly filter out the node that does not belonging to 11 types of UI components. This is because, as discussed above, most mobile applications have GUI design consistency in different user interfaces within one application. Therefore, the changes in other nodes may not affect the screenshots. For example, in Figure 3.4(c), we can see that the position and size in image view changes, however, the screenshots should be regarded as similar and need to be removed, as the perspective of our components

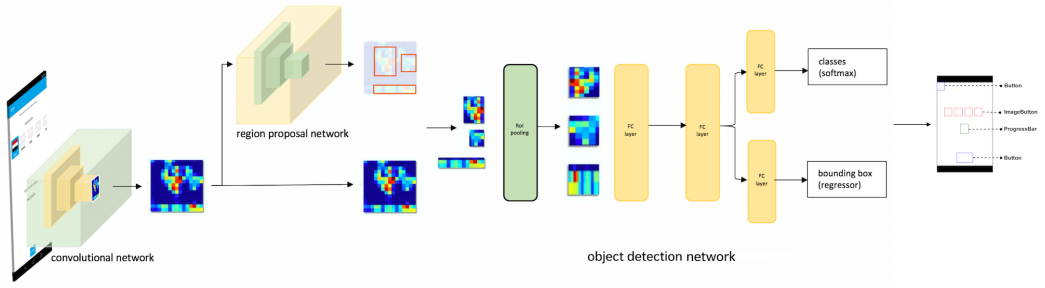


Figure 3.5: The architecture of Faster RCNN [Ren et al., 2015]

is the same. Then, we apply Breath-First [Cormen et al., 2009] search algorithm to construct a string which are formed by components type and its coordinate. For example, the XML tree format of GUI screenshot is shown in the lower left corner in Figure 3.3. Red nodes indicate the remaining nodes after filtering. The final string is *"CheckBox[21,499][191,542]SeekBar[21,456][779,499]Button[26,209][269,273]..."*. To calculate the string sequence similarity, we use Levenshtein distance[Yarkoni et al., 2008]. If the two GUI screenshots reached the similarity threshold, then they will be regarded as the same UI state and one of them will be removed. Finally, we obtain 43,629 screenshots that are relatively unique.

3.2.3 UI component wirification

Our design wirification task is an immensely important research topic in computer vision domain, as known as "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 [Girshick, 2015; Ren et al., 2015; He et al., 2017]. In our work, we adopt the Faster-RCNN model [Ren et al., 2015], which is a state-of-the-art object detection model on several fields. Figure 3.5 presents the overall architecture of the model. The model uses a Convolutional Neural Network [LeCun et al., 1998; Krizhevsky et al., 2012] to extract image features from the input GUI screenshot. It then use a region proposal network (RPN) [Ren et al., 2015] to generate region proposals that likely contain some object of interest (as opposed to only background). These regional proposal are called Region of Interest (ROI) [Brett et al., 2002]. Finally, it uses an object detection network that predict object classification scores for the region proposals and determines the object bounding box.

3.2.4 Post-processing GUI Components

We further post-processing these GUI components obtained through GUI component wirification in Section 3.2.3. To improve the diversity of design gallery, we remove

many duplicated or very similar GUI components by a composite measure on both structure (structural similarity index [Wang and Bovik, 2002; Wang et al., 2004]) and color layout information (Color Histogram [Uijlings et al., 2013]). 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. In the following subsections, we will discuss how those values be expressed.

3.2.4.1 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 (Figure 3.4(a,b,c)). 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. Image similarity is a tremendously important field in computer vision. Many works indicates pHash [Kozat et al., 2004] can reach state-of-the-art measurement. However, it is not a reliable method in our context. The core techniques of pHash is to transform image to "fingerprint" based on identifying salient features in the image. A shifting leads to different "fingerprint", resulting in poor performance. And laterally shifting components appear abundantly in our database. Furthermore, pHash is not sensitive to color. Therefore, we introduce a hybrid method that leverages both the structure and color information within the component to determine how similar two GUI components are.

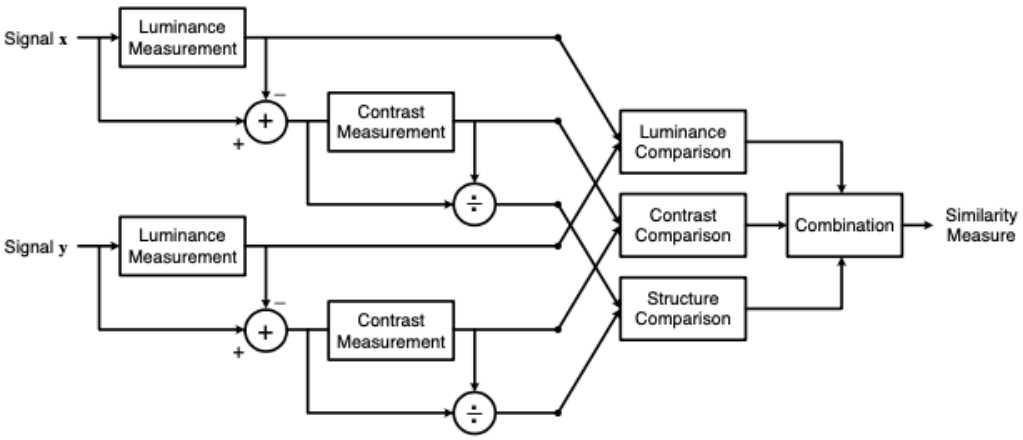


Figure 3.6: Illustration of structural similarity (SSIM) index [Hore and Ziou, 2010] process.

Structure Similarity Mean Squared Error (MSE) [Lehmann and Casella, 2006] is a broadly recognized and accepted to measure the structure of image. Recent research work [Wang and Bovik, 2009] indicates its weakness in some applications such as

visual perception of images. We adopt an advanced method - structural similarity index (SSIM) [Wang and Bovik, 2002; Wang et al., 2004] algorithm. Structural similarity measure can be written as a composite measure of three factors: luminance, contrast and structure. Each factor is relatively independent. For example, the change on luminance does not affect the contrast of image. Therefore, we calculate each factor to yield an overall measurement. An illustration is shown in Figure 3.6. First we calculate luminance comparison part, it can be calculated by mean intensity:

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \quad \text{where} \quad \mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3.1}$$

where C_1 is to prevent the denominator is zero.

Second, we using the standard deviation as an estimation of the image contrast.

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \quad \text{where} \quad \sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2} \tag{3.2}$$

Third, the image is normalized to an unit standard deviation, in terms of the structure of the image. Hence, the structure comparison can be simplified as:

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{3.3}$$

And then takes the combination as the final similarity measurement of structure:

$$SSIM(x,y) = l(x,y) * c(x,y) * s(x,y) \tag{3.4}$$

Color Similarity We adopt one-dimensional colour histograms [Uijlings et al., 2013] to measure color similarity. It transform each channel within the image into histogram using 25 bins (i.e., $C_R = \{c_R^1, c_R^2, ..., c_R^{25}\}$), which performs well in previous work [Uijlings et al., 2013]. Then, we calculate the distance intersection:

$$S_{color}(x_{<r,g,b>}, y_{<r,g,b>}) = \sum_{k=1}^{N}(min(c^k_{x_{<r,g,b>}}, c^k_{y_{<r,g,b>}})) \tag{3.5}$$

The combination of structural and color similarity yields overall final similarity. Based on 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.

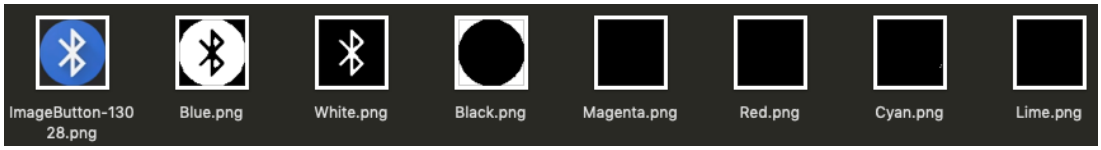


Figure 3.7: An example of determining the primary color of GUI component by HSV mask

3.2.4.2 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. Commonly RGB based color space is regarded as the default starting point for color detection [Dony and Wesolkowski, 1999]. However, recent research shows that RGB color space is not an appropriate approach to detect and analyse color because it mixes of color (chrominance) and intensity (luminance) information, instead of uniform characteristics [Schwarz et al., 1987; Singh et al., 2003]. Note that we adopt color histogram in Section 3.2.4.1 to measure similarity because the mixture does not have a significant impact on the color layout information.

To that end, we adopt an advanced method - HSV color space for color detection which is a proven color discrimination approach in computer vision community [Shaik et al., 2015]. We first makes a transformation from RGB color to HSV color space. 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. 3.7. 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 Figure 3.7).

3.2.4.3 Identifying Font of UI Components

User Interface is regarded as the primary point of interaction between the user and the app. Therefore it is critical to design a user interface adjusted to the requirements of the target group. Rather than some obvious adjusting features (i.e., position, size), font is a feature that can easily be ignored. However, it has a great potential effect on the readability of UI [Boll and Brune, 2015]. A suitable font results in better interaction between the user and the application. *Sans Serif* font is preferred in most applications because of its good readability. However, *Sans Serif* may not consistent with the category of applications. For example, in "Game" applications, designers

prefer using *Gtek Technology* fonts than *Sans Serif* fonts, because *Sans Serif* portray tradition, sophistication and a formal tone, but are not suitable in leisure applications. Instead, *Gtek Technology* fonts portray relax and gaming style.

In order to identify the font in UI components, we attempt several algorithm such as 100-font classifier [Baird and Nagy, 1994] and font identification by clusters [Khoubyari and Hull, 1996]. However, due to the limitation of font resources, we adopt to use Myfonts tool [wha, 2019] to collect the font information that is arguably the most well-known of all the free font finder apps powered by the world's largest collection of fonts.

3.3 Results

In this section, we will first analyse the performance of our model (Section 3.2.3), as well as discussing our finding. Then, we compare the derivations of our model with several basic baselines including deep-learning and machine-learning. Finally, we express the model behaviours on some example UIs to see how it performs on the real world data.

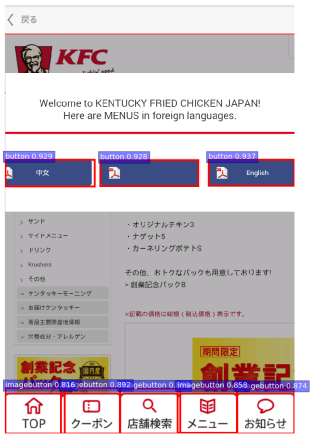
3.3.1 Evaluation Metric

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.

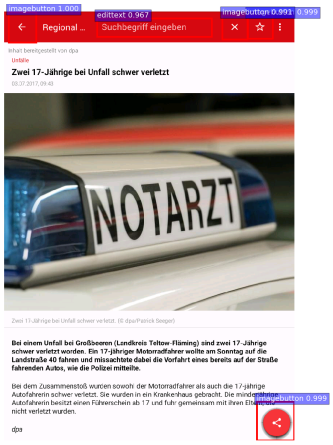
The three principal metrics used for evaluating and comparing the performance are precision, recall and mean Average Precision (mAP). Precision of each class is obtained through dividing the cumulative sum of True-Positive by the total amount of True-Positive and False-Positive for an individual class. 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.

3.3.2 Evaluation with Screenshots

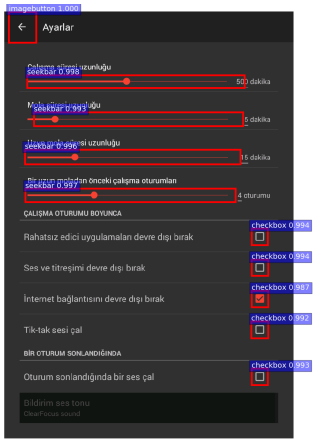
Among all 68,702 annotated runtime GUI screenshots collected through automated GUI exploration, we perform a 90/10 train/test split (i.e., 61,832 for training the UI component wirification model and 6,870 for testing the trained model). For the 6,870 annotated GUI screenshots in the testing dataset, the overall performance of our model is 0.69 for precision, 0.51 for recall and 0.57 for mAP. Figure 3.8 shows some



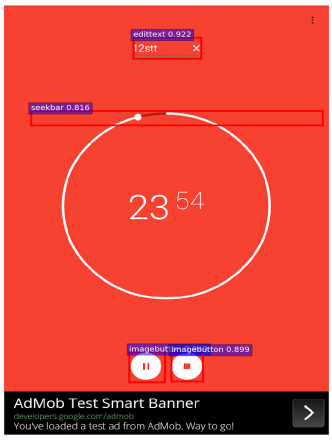
(a) Components with texts in different languages (accurately detected)



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

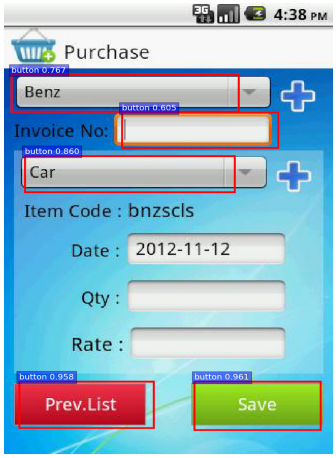


(c) Many components (accurately detected)



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

(e) Subtle difference between image, image button and button (object classification is inaccurate)



(f) Imbalanced class (object classification is inaccurate)

Figure 3.8: Results of GUI component wirification in complex GUI screenshots

```
java.lang.Object
↳ android.view.View
   ↳ android.widget.TextView
      ↳ android.widget.Button
```

(a) Button extends chain

```
java.lang.Object
↳ android.view.View
   ↳ android.widget.ImageView
      ↳ android.widget.ImageButton
```

(b) Image Button extends chain

Figure 3.9: Documentation of Android widgets

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, Figure 3.8(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 Figure 3.8(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 Figure 3.8(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 four reasons for the incorrect cases. First, sometimes our model cannot identify some specially designed UI components like the circle-shape seek bar in Figure 3.8(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. 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. For example, in Figure 3.8(e), although we detect the red cross button, but it is a wrong prediction due to the ground truth is a "clickable" image. Third, the example of Figure 3.8(e) also shows us another problem. For example, the red cross button in Figure 3.8(e) is classified as button, it can be image button as well. Although button and image button has a clear definition in Android Source tree level as shown in Figure 3.9 (i.e., you can't call setText for ImageButton, you can with a regular button), they derive from view. Thus, it is difficult to distinguish them in image perception. Fourth, our dataset has a potential problem on class distribution. The proportion of button and image button in our dataset is 38.17% and 42.89% respectively. Because button and image button are in the majority, it is highly possible overfit (i.e., misrecognise other widgets as button or image button). For example, in Figure 3.8(f), we can see that the model recognise spinner as button by shortening the bound and detect inputfield as button. This causes wrong detection.

3.4 Gallery system design and construction

We compile a large-scale gallery of GUI component designs (together with the original whole GUIs) obtained in Section 3.2.3. To further extend our gallery, we wirify GUI component designs in Rico [Deka et al., 2017]. A distribution of component class is shown in Table 3.1. Then, we implement a web-based search interface

	Size	Proportion
Button	13,970	43.28%
ImageButton	8,182	25.35%
CheckBox	1,838	5.69%
RadioButton	2,803	8.68%
Switch	614	1.90%
ToggleButton	1,045	3.24%
ProgressBar	707	2.19%
RatingBar	392	1.21%
SeekBar	728	2.26%
Spinner	1,966	6.09%
Chronometer	33	0.10%

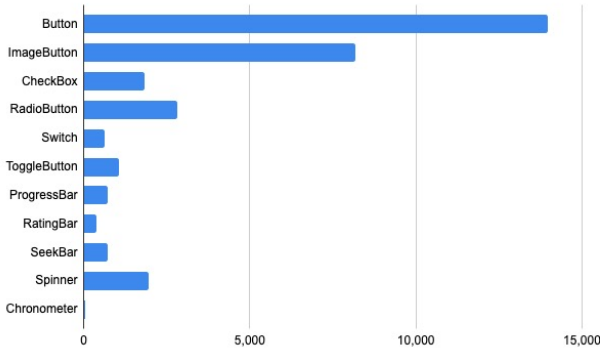


Table 3.1: A distribution of the wirified component.

(http://mui-collection.herokuapp.com/) that supports multi-faceted GUI component search by multi-dimensional component metada (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 summarize 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 Figure 3.15) for designers to visually compare GUI components side by side. For the copyright, the main target of our work is to help GUI designers understand and get inspiration from existing apps, instead of directly distributing any part of the code for developing apps for commercial purpose. The approach in our work can be regarded as a kind of reverse engineering process which is allowed by both the US law[1] and Europe Law[2] which explicitly offer users the right to decompile in order to achieve interoperability. In details, Section 103(f) of the Digital Millennium Copyright Act (DMCA) (17 USC Âğ 1201 (f) - Reverse Engineering) in the U.S. law specifically states that it is legal to reverse engineer and circumvent the protection to achieve interoperability between computer programs (such as information transfer between applications).

3.4.1 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

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

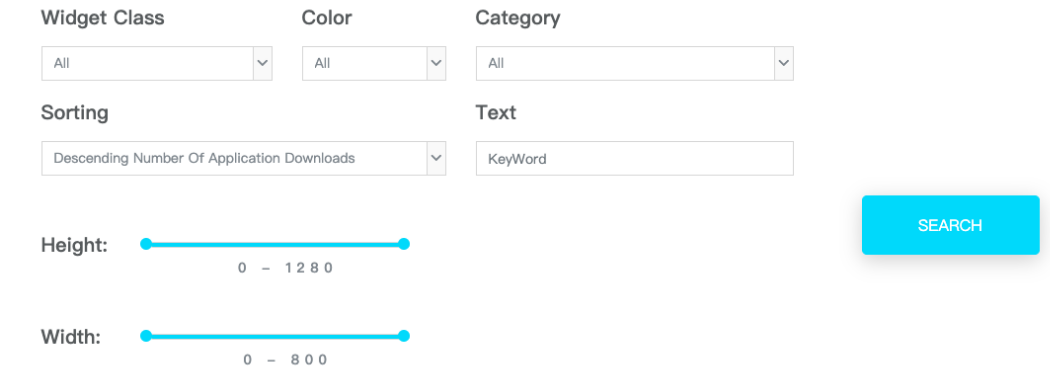


Figure 3.10: Multi-faceted design search inspired by booking and shopping websites

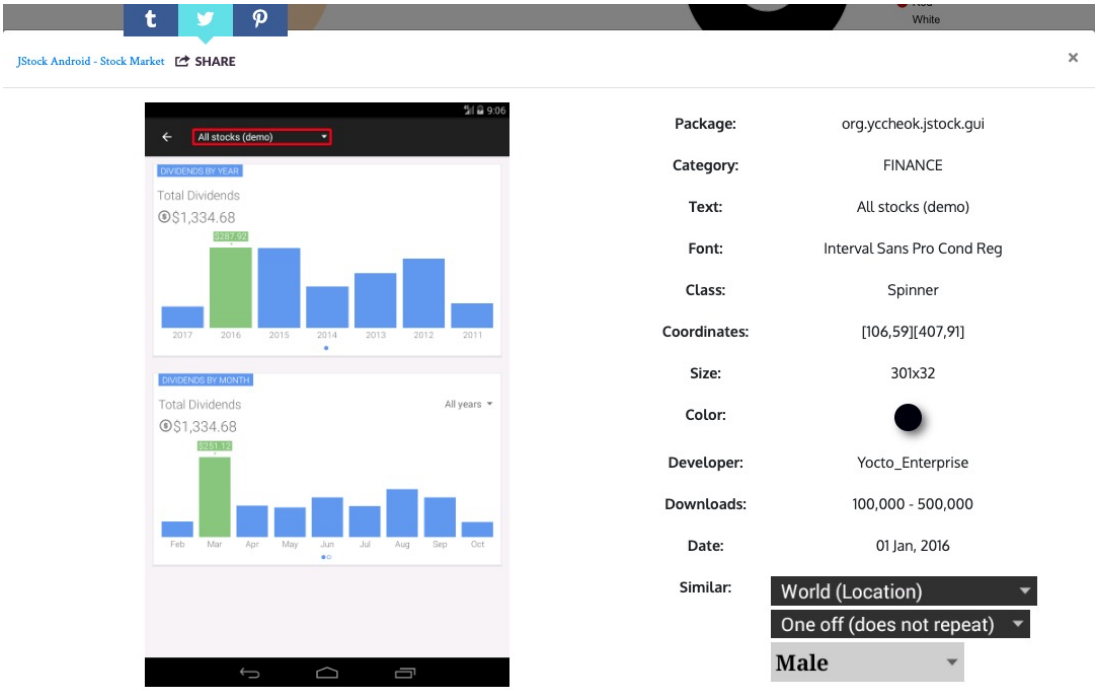


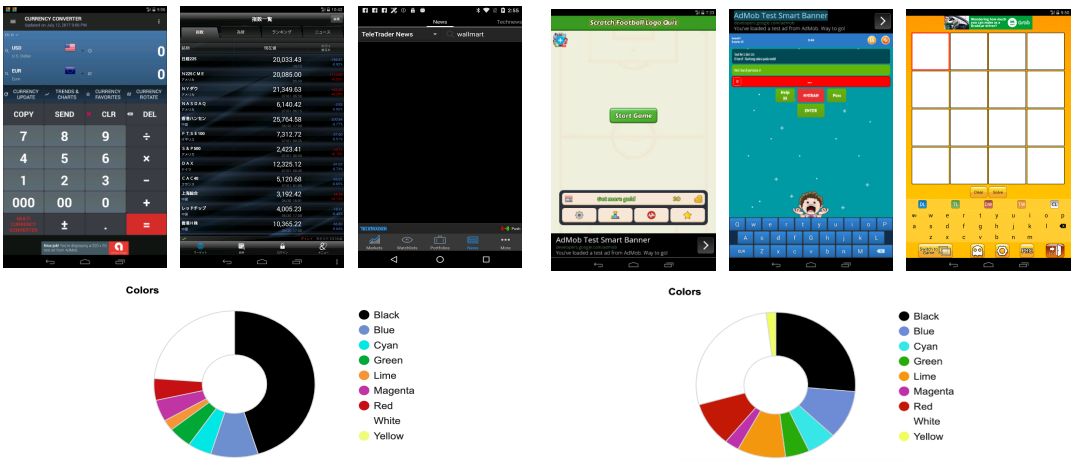
Figure 3.11: A screenshot of our gallery application

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. An illustration of our multi-faceted search interface is shown in Figure 3.10. In our prototype, we support 11 types of GUI components (see Section 3.2) The component primary color is determined as described in Section 3.2.4.2. 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.

Users can freely combine the five facets in their search. Some detailed usage scenarios are demonstrated in Section 3.5.1. 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. 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 (see in Figure 3.11).

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. Therefore, the system carries better performance and load time. This core technique of this strategy is an event listener which listens to the events in the browser such as scrolling, resizing, etc. After we recognize a change, we determine the space in the browser and calculate the number of images that should become visible on the screen using window height, current document top position, image's top offset, etc. Finally, we trigger them to load accordingly.



(a) Finance application screenshots and color demographics

(b) Entertainment application screenshots and color demographics

Figure 3.12: Color demographics - finance versus entertainment

3.4.2 Design Demographics

In addition to view individual GUI designs, designers also need to understand the overall design landscape [Chen et al., 2016; Chen and Xing, 2016] 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. For example, Figure 3.12 summarizes the distributions of component primary colors in finance applications and entertainment applications, respectively. The statistics show that GUI components of finance applications mainly use black or dark color[3], while entertainment applications use more diverse and vivid colors. Some usage scenarios are demonstrated in Section 3.5.2.

3.4.3 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 demon-

[3]See an interesting discussion on the color psychology for finance applications at https://ux.stackexchange.com/questions/25167/why-do-most-financial-apps-use-a-black-or-dark-background

strated in Section 3.5.3.

3.4.4 Design Sharing

For each GUI component which designers are interested in, we also provide the sharing function. For example, designers can come to our site, and share the components that they are interested in by simply clicking the sharing button in our site to their Twitter[Twitter, 2019] and Facebook[Facebook, 2019] as shown in Figure 3.11. In addition, we also allow the designers to pin the components into their Pinterest[Pinterest, 2018] for collecting the attracting design, so that they can further discuss the design with their colleagues for reshaping their own design. Thus, those social elements of our site can facilitate the collaboration among designers.

3.5 Informal Feedback from Designers

To understand how the design gallery 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.

For D1 to D5, We demonstrated our gallery 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. 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

Figure 3.13: Two attracting game buttons

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-stream 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.

3.5.1 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 3.13. 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 3.13). 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.

Figure 3.14: Buttons from social apps and corresponding demographics

3.5.2 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, she searched the GUI components by social application category. Apart from the buttons as seen in Fig 3.14, 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.

3.5.3 Task 3: Design "Comparison Shopping"

D7 was working on creating the UI component design system [Uplab, 2018; Baek and Bae, 2016] 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. 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 3.15. 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.

3.5.4 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.

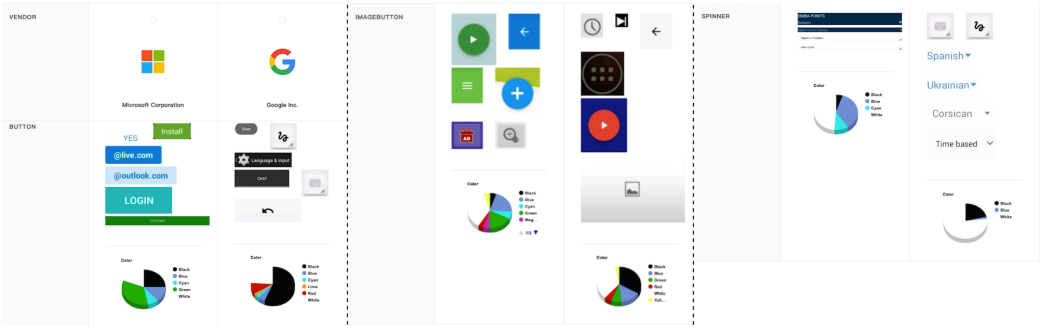


Figure 3.15: Comparison of components between Google and Microsoft

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.

3.6 Implications

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

3.6.1 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 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".

3.6.2 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 [Chen et al., 2017a, 2018a], 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.

3.6.3 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 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.

Chapter 4

Conclusion

In this project, we present a deep-learning based approach for harvesting crowdsourcing GUI design resources, which allows us to build a large scale GUI and GUI component design gallery. On the one hand, albeit a large number of GUI designs online with human pre-described tags, many GUIs are poorly described, resulting in the incompleteness on retrieval. Based on the systematical analysis of the existing large-scale GUI designs, we carry out a formal empirical study to construct a vocabulary of UI semantics. To have a better design retrieval, we present a hybrid model capturing both the visual and textual information to automatically complete the description of GUI on tags. On the other hand, GUI component-oriented search is typically unsupported in the online sharing platforms. The core techniques to turn crowdsourcing GUI design resources into a GUI components gallery include a reverse-engineering technique to collect app runtime GUI screenshots and an object-detection 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; not only design holism but also design granularity.

4.1 Future Work

In the future, we will keep improving our work. Despite our design gallery focuses on the a large scale GUI design in multi-aspects including artistic, practical, holistic and component-oriented, it is still critical to integrate as a whole. We will separate the future task into three directions: search interface, data mining and design synthesis.

4.1.1 Search Interface

First, either the multi-faceted or tagging-based search requires a selection or textual query. In the future, we will develop a more advanced tool to achieve a better user experience. For example, our tool may take the component image as a query, and not only return similar components, but also other-type components whose design style can fit into the query's. Although we try to reuse similarity measurement

in Section 3.2.4.1, due to the limitation of engineering effort (i.e., OpenCV is not fully supported in web development), we adopt a simple similarity measurement pHash [Kozat et al., 2004]. However, it does not perform well because of several reasons. First, color histogram is an unreliable features in our case as discussed in Section 3.2.4.1. Second, the class of image is unknown. For example, the user inputs a RatingBar to search any similar image. Since the lack of component classifiers, we need to check all the components. However, it leads to two problems. One is expensive and time-consuming, as we have 34,913 components in the database. The other one is inaccuracy of the result, as different type of components may look similar (i.e., progress bar and circle-like image button). Thus, a component classifier is one crucial work in the future.

4.1.2 Data Mining

The feedback from designers will be taken take into account to improve our search interface. For example, as mentioned in Section 3.5.4, 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. As reported in Section 2.5.4, we will leverage the video analysis method to explore the dynamic animation design resource. Users will get animation design in tagging based search for more inspirations. Furthermore, feedbacks also report the limitation of the retrieved result. We will extend our approach into other platforms including website, IOS, or other IOT (Internet of Things) devices for supporting broader UI designs.

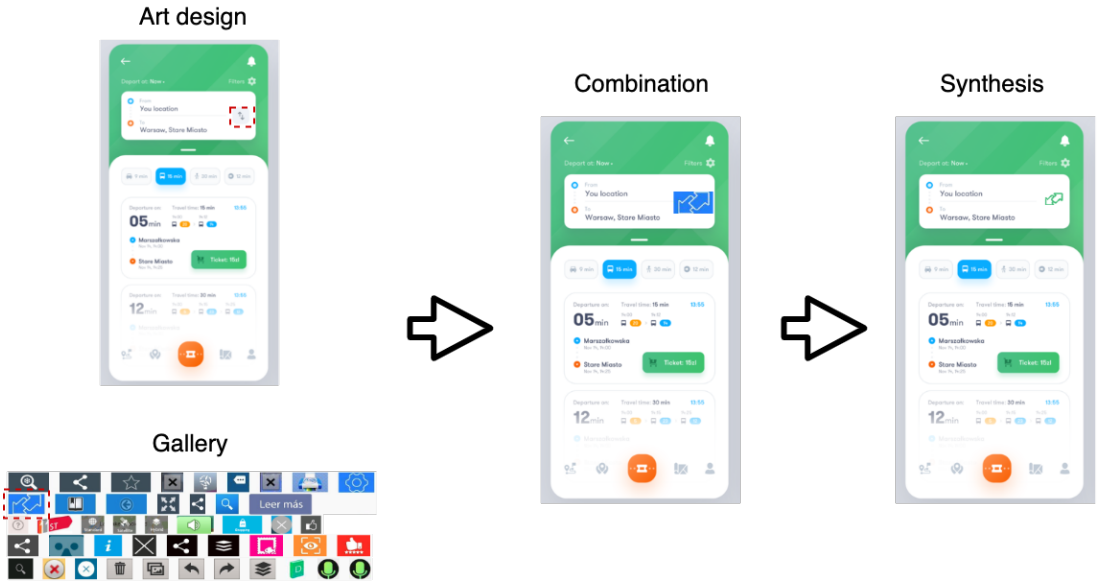


Figure 4.1: An usage scenario for design synthesis process.

4.1.3 Design Synthesis

A usage scenario is shown in Figure 4.1. The user can first search a desired whole UI art design. Then, the user may have an interesting practical component design that he wants to apply to the art design. Although the whole UI art design and the practical component design look good in their context, the combination looks odd (i.e., the size is consistent, the color is consistent). Therefore, a design composition synthesis (i.e., critique) tool is required. We adopt a DCGAN [Radford et al., 2015; Iizuka et al., 2017] for UI design critique. Rather than training the whole UI design, we crop a 256×256 components related area, see examples in Table 4.1. This is because, the origin screenshot is in high dimension (800×1280), resulting in the large parameters of the model and causing the training infeasible [Hinton and Salakhutdinov, 2006]. While it sounds reasonable to train a 256×256 instance by rescaling since the training instance captures all the hierarchy of image and meanwhile the training instance allows multi-mask area to achieve efficiency, the size of our mask is relatively small. In general, single component only accounts for 5% of the entire design. Therefore, it may cause two problems. First, the problem of vanishing gradient in deep neural network [Hochreiter, 1998]. Although it is resolved by Residual Connection [He et al., 2016] in many deep neural network problems such as ResNet [He et al., 2016], it is not explored deeply in GANs research community. Second, the problem of imprecise component design. Once the origin design is rescaled, the component design becomes obscure. It becomes even worse for the generated design. Thus, to overcome these problems, we make a trade-off on the hierarchy of image. We only train the hierarchy around components. Note that we keep the component in the middle of the area to capture the hierarchy in all directions. In order to correct the component background color, we make a mask (i.e., randomly change the component background color) on the components and feed it to train. Table 4.1 shows some results generated by the model. In some cases, while the generated design is different to the ground truth, it may looks reasonable. We determine these designs as serendipitous designs. Therefore, an improvement on the model and the output discrimination will be a part of future work.

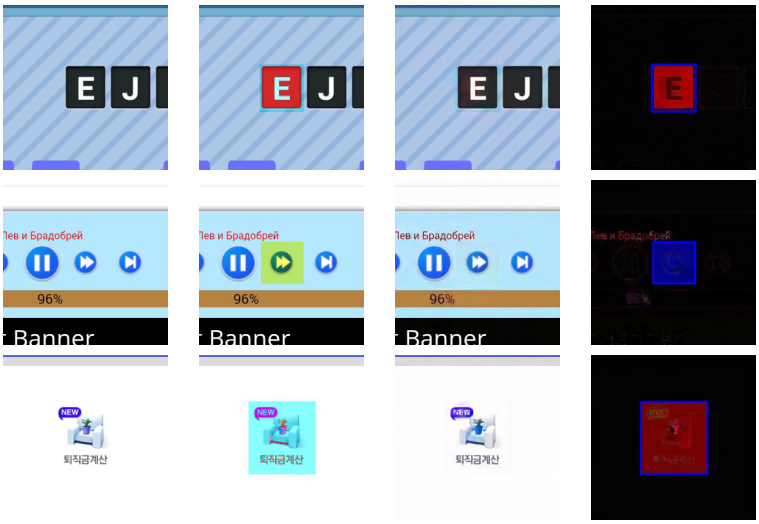
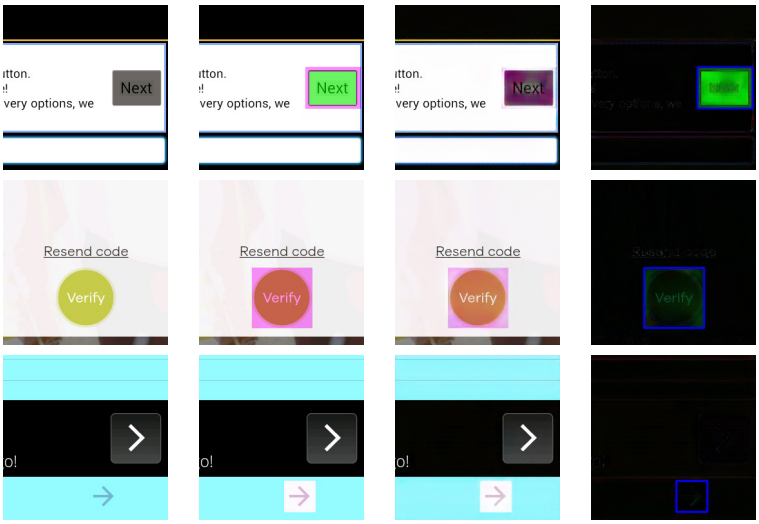
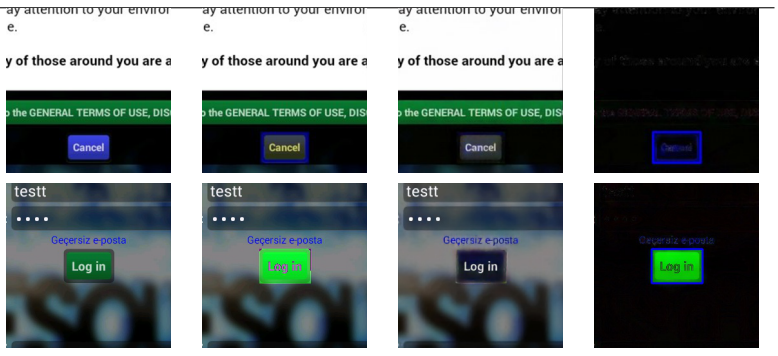
	Original	Input	Output	Modified
Good result				
Bad result				
Serendipitous result				

Table 4.1: An example of UI designs generated by DCGAN [Radford et al., 2015].

Bibliography

2015. 2015 mobile trends report. https://resources.axway.com/mobile-app-dev/report-2015-mobile-trends-report. Accessed: 2019-10-12. (cited on page 6)

2017. Essential design principles. https://developer.apple.com/videos/play/wwdc2017/802/. (cited on page 2)

2018. Ui automator. https://developer.android.com/training/testing/ui-automator. Accessed: 2018-04-25. (cited on page 40)

2019. https://dumps.wikimedia.org/. (cited on page 20)

2019. https://www.myfonts.com/WhatTheFont. (cited on page 46)

Agrawal, R.; Srikant, R.; et al., 1994. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 487–499. (cited on page 13)

Alharbi, K. and Yeh, T., 2015. Collect, decompile, extract, stats, and diff: Mining design pattern changes in android apps. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 515–524. ACM. (cited on page 10)

Amalfitano, D.; Fasolino, A. R.; and Tramontana, P., 2011. A gui crawling-based technique for android mobile application testing. In *2011 IEEE fourth international conference on software testing, verification and validation workshops*, 252–261. IEEE. (cited on page 11)

Anderson, J.; McRee, J.; Wilson, R.; et al., 2010. *Effective UI: The art of building great user experience in software*. " O'Reilly Media, Inc.". (cited on page 2)

Axure, 2018. Axure widgets. https://axurewidgets.com/download-axure-widgets/. Accessed 2018. (cited on pages 3 and 36)

Baek, Y.-M. and Bae, D.-H., 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 (Singapore, Singapore, 2016), 238–249. ACM, New York, NY, USA. doi:10.1145/2970276.2970313. http://doi.acm.org/10.1145/2970276.2970313. (cited on page 55)

Baird, H. S. and Nagy, G., 1994. Self-correcting 100-font classifier. In *Document Recognition*, vol. 2181, 106–115. International Society for Optics and Photonics. (cited on page 46)

Bastian, M.; Heymann, S.; Jacomy, M.; et al., 2009. Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8 (2009), 361–362. (cited on page 14)

Behance, 2005. Behance :: Best of behance. https://www.behance.net/. (cited on pages xiii and 2)

Behrang, F.; Reiss, S. P.; and Orso, A., 2018. Guifetch: supporting app design and development through gui search. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, 236–246. ACM. (cited on pages 10 and 37)

Bernal-Cardenas, C.; Moran, K.; Tufano, M.; Liu, Z.; Nan, L.; Shi, Z.; and Poshyvanyk, D., 2019. Guigle: A gui search engine for android apps. *arXiv preprint arXiv:1901.00891*, (2019). (cited on page 3)

binti Ayob, N. Z.; Hussin, A. R. C.; and Dahlan, H. M., 2009. Three layers design guideline for mobile application. In *2009 International Conference on Information Management and Engineering*, 427–431. IEEE. (cited on page 2)

Blondel, V. D.; Guillaume, J.-L.; Lambiotte, R.; and Lefebvre, E., 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008, 10 (2008), P10008. (cited on page 14)

Boll, F. and Brune, P., 2015. User interfaces with a touch of grey?–towards a specific ui design for people in the transition age. *Procedia Computer Science*, 63 (2015), 511–516. (cited on page 45)

Borning, A. and Duisberg, R., 1986. Constraint-based tools for building user interfaces. *ACM Transactions on Graphics (TOG)*, 5, 4 (1986), 345–374. (cited on page 6)

Brett, M.; Anton, J.-L.; Valabregue, R.; Poline, J.-B.; et al., 2002. Region of interest analysis using an spm toolbox. In *8th international conference on functional mapping of the human brain*, vol. 16, 497. Sendai, Japan. (cited on page 42)

Chen, C.; Chen, X.; Sun, J.; Xing, Z.; and Li, G., 2018a. 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. (cited on page 57)

Chen, C.; Su, T.; Meng, G.; Xing, Z.; and Liu, Y., 2018b. 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). (cited on page 37)

Chen, C. and Xing, Z., 2016. Mining technology landscape from stack overflow. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 14. ACM. (cited on page 52)

Chen, C.; Xing, Z.; and Han, L., 2016. Techland: Assisting technology landscape inquiries with insights from stack overflow. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 356–366. IEEE. (cited on page 52)

Chen, C.; Xing, Z.; and Liu, Y., 2017a. 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. (cited on page 57)

Chen, C.; Xing, Z.; and Wang, X., 2017b. Unsupervised software-specific morphological forms inference from informal discussions. In *Proceedings of the 39th International Conference on Software Engineering*, 450–461. IEEE Press. (cited on page 17)

Chen, M.; Zheng, A.; and Weinberger, K., 2013. Fast image tagging. In *International conference on machine learning*, 1274–1282. (cited on page 11)

Chen, S.; Fan, L.; Chen, C.; Su, T.; Li, W.; Liu, Y.; and Xu, L., 2019. Storydroid: Automated generation of storyboard for android apps. In *Proceedings of the 41st International Conference on Software Engineering*. ACM. (cited on pages 10 and 36)

Clifton, I. G., 2015. *Android User Interface Design: Implementing Material Design for Developers*. Addison-Wesley Professional. (cited on page 2)

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C., 2009. *Introduction to algorithms*. MIT press. (cited on page 42)

Cortes, C. and Vapnik, V., 1995. Support-vector networks. *Machine learning*, 20, 3 (1995), 273–297. (cited on page 25)

Dasdan, A., 2011. Sharing tagged data on the internet. US Patent 7,953,775. (cited on page 3)

Deka, B.; Huang, Z.; Franzen, C.; Hibschman, J.; Afergan, D.; Li, Y.; Nichols, J.; and Kumar, R., 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*, 845–854. ACM. (cited on pages 5, 10, 36, and 48)

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L., 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee. (cited on page 20)

Dony, R. and Wesolkowski, S., 1999. Edge detection on color images using rgb vector angles. In *Engineering Solutions for the Next Millennium. 1999 IEEE Canadian Conference on Electrical and Computer Engineering (Cat. No. 99TH8411)*, vol. 2, 687–692. IEEE. (cited on page 45)

Dribbble, 2010. Dribbble - discover the world's top designers & creatives. https://dribbble.com/. (cited on pages xiii, xv, 2, 3, 9, 11, 32, and 35)

Duygulu, P.; Barnard, K.; de Freitas, J. F.; and Forsyth, D. A., 2002. Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary. In *European conference on computer vision*, 97–112. Springer. (cited on page 10)

Facebook, 2019. https://facebook.com. (cited on page 53)

Finlayson, G. D.; Schiele, B.; and Crowley, J. L., 1998. Comprehensive colour image normalization. In *European conference on computer vision*, 475–490. Springer. (cited on page 19)

Fischer, M.; Yang, R. R.; and Lam, M. S., 2018. Imaginenet: Style transfer from fine art to graphical user interfaces. (2018). (cited on page 10)

Fitzmaurice, G. W.; Balakrishnan, R.; Kurtenbach, G.; and Buxton, B., 1999. An exploration into supporting artwork orientation in the user interface. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 167–174. ACM. (cited on page 2)

Fu, B.; Lin, J.; Li, L.; Faloutsos, C.; Hong, J.; and Sadeh, N., 2013. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 1276–1284. ACM. (cited on page 10)

Fu, X., 2010. Mobile phone ui design principles in the design of human-machine interaction design. In *2010 IEEE 11th International Conference on Computer-Aided Industrial Design & Conceptual Design 1*, vol. 1, 697–701. IEEE. (cited on page 6)

Galitz, W. O., 2007. *The essential guide to user interface design: an introduction to GUI design principles and techniques*. John Wiley & Sons. (cited on pages 2 and 6)

Girshick, R., 2015. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, (2015). (cited on page 42)

Google, 2019. https://play.google.com/store/apps. (cited on page 4)

GraphicBurger, 2013. Elven iphone app ui kit. https://graphicburger.com/elven-iphone-app-ui-kit/. Accessed: 2018-04-25. (cited on pages xiii, 2, 3, 11, and 35)

Harris, D. and Harris, S., 2010. *Digital design and computer architecture*. Morgan Kaufmann. (cited on page 20)

He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R., 2017. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2980–2988. IEEE. (cited on page 42)

He, K.; Zhang, X.; Ren, S.; and Sun, J., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778. (cited on pages 20 and 61)

Heydon, A. and Najork, M., 1999. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2, 4 (1999), 219–229. (cited on page 11)

Hinton, G. E. and Salakhutdinov, R. R., 2006. Reducing the dimensionality of data with neural networks. *science*, 313, 5786 (2006), 504–507. (cited on page 61)

Hochreiter, S., 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6, 02 (1998), 107–116. (cited on page 61)

Hong, Y. W. B., 2011. Matters of design. In *Commun. ACM*. Citeseer. (cited on page 6)

Hore, A. and Ziou, D., 2010. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, 2366–2369. IEEE. (cited on pages xiii and 43)

Iizuka, S.; Simo-Serra, E.; and Ishikawa, H., 2017. Globally and locally consistent image completion. *ACM Transactions on Graphics (ToG)*, 36, 4 (2017), 107. (cited on page 61)

Inokuchi, A.; Washio, T.; and Motoda, H., 2000. An apriori-based algorithm for mining frequent substructures from graph data. In *European conference on principles of data mining and knowledge discovery*, 13–23. Springer. (cited on pages xiii and 13)

Inspired-ui, 2018. Inspired-ui. http://inspired-ui.com/. Accessed 2018. (cited on pages 3 and 36)

Jahanian, A.; Isola, P.; and Wei, D., 2017a. Mining visual evolution in 21 years of web design. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 2676–2682. ACM. (cited on page 10)

Jahanian, A.; Keshvari, S.; Vishwanathan, S.; and Allebach, J. P., 2017b. Colors–messengers of concepts: Visual design mining for learning color semantics. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 24, 1 (2017), 2. (cited on page 10)

Jansen, B. J., 1998. The graphical user interface. *ACM SIGCHI Bulletin*, 30, 2 (1998), 22–26. (cited on page 1)

Jeon, J.; Lavrenko, V.; and Manmatha, R., 2003. Automatic image annotation and retrieval using cross-media relevance models. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, 119–126. ACM. (cited on page 10)

Jo, I. and Jung, I. Y., 2016. Smart learning of logo detection for mobile phone applications. *Multimedia Tools and Applications; Dordrecht*, , 13211-13233 (2016). (cited on page 37)

Kalayeh, M. M.; Idrees, H.; and Shah, M., 2014. Nmf-knn: Image annotation using weighted multi-view non-negative matrix factorization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 184–191. (cited on page 11)

Khoubyari, S. and Hull, J. J., 1996. Font and function word identification in document recognition. *Computer Vision and Image Understanding*, 63, 1 (1996), 66–74. (cited on page 46)

Kozat, S. S.; Venkatesan, R.; and Mihçak, M. K., 2004. Robust perceptual image hashing via matrix invariants. In *2004 International Conference on Image Processing, 2004. ICIP'04.*, vol. 5, 3443–3446. IEEE. (cited on pages 43 and 60)

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105. (cited on pages 19 and 42)

Kumar, R.; Satyanarayan, A.; Torres, C.; Lim, M.; Ahmad, S.; Klemmer, S. R.; and Talton, J. O., 2013. Webzeitgeist: design mining the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 3083–3092. ACM. (cited on page 36)

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86, 11 (1998), 2278–2324. (cited on pages 19 and 42)

Lehmann, E. L. and Casella, G., 2006. *Theory of point estimation*. Springer Science & Business Media. (cited on page 43)

Levenshtein, V. I., 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, vol. 10, 707–710. (cited on page 17)

Li, J. and Wang, J. Z., 2003. Automatic linguistic indexing of pictures by a statistical modeling approach. *IEEE Transactions on pattern analysis and machine intelligence*, 25, 9 (2003), 1075–1088. (cited on page 10)

Liu, T. F.; Craft, M.; Situ, J.; Yumer, E.; Mech, R.; and Kumar, R., 2018. Learning design semantics for mobile apps. In *The 31st Annual ACM Symposium on User Interface Software and Technology*, 569–579. ACM. (cited on page 10)

Loper, E. and Bird, S., 2002. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*, (2002). (cited on page 17)

Martin, W.; Sarro, F.; Jia, Y.; Zhang, Y.; and Harman, M., 2017. A survey of app store analysis for software engineering. *IEEE transactions on software engineering*, 43, 9 (2017), 817–847. (cited on page 10)

Material, G., 2014. Design - material design. https://material.io/design/. (cited on page 15)

Meier, S.; Heidmann, F.; and Thom, A., 2014. A comparison of location search ui patterns on mobile devices. *MobileHCI '14*, , 465-470 (2014). (cited on page 37)

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J., 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, (2013). (cited on page 20)

Mikolov, T.; Karafiát, M.; Burget, L.; Černockỳ, J.; and Khudanpur, S., 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*. (cited on page 20)

Mikolov, T.; Le, Q. V.; and Sutskever, I., 2013b. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, (2013). (cited on page 25)

Moran, K.; Bernal-Cárdenas, C.; Curcio, M.; Bonett, R.; and Poshyvanyk, D., 2018. Machine learning-based prototyping of graphical user interfaces for mobile apps. *arXiv preprint arXiv:1802.02312*, (2018). (cited on page 37)

Mori, Y.; Takahashi, H.; and Oka, R., 1999. Image-to-word transformation based on dividing and vector quantizing images with words. In *First international workshop on multimedia intelligent storage and retrieval management*, 1–9. Citeseer. (cited on page 10)

Muhammad, B. and Abu-Bakar, S. A. R., 2015. A hybrid skin color detection using hsv and ycgcr color space for face detection. In *2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, 95–98. IEEE. (cited on page 5)

Najork, M. and Wiener, J. L., 2001. Breadth-first crawling yields high-quality pages. In *Proceedings of the 10th international conference on World Wide Web*, 114–118. ACM. (cited on pages xiii, 11, and 12)

Neil, T., 2014. *Mobile design pattern gallery: UI patterns for smartphone apps*. " O'Reilly Media, Inc.". (cited on page 15)

Nguyen, T. A. and Csallner, C., 2015. Reverse engineering mobile application user interfaces with remaui (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 248–259. IEEE. (cited on page 37)

Norman, D. A., 2004. *Emotional design: Why we love (or hate) everyday things*. Basic Civitas Books. (cited on page 6)

Parker, R.; Graff, D.; Kong, J.; Chen, K.; and Maeda, K., 2011. English gigaword fifth edition, linguistic data consortium. *Google Scholar*, (2011). (cited on page 20)

Pedrycz, W. and Chen, S.-M., 2011. *Granular computing and intelligent systems: design with information granules of higher order and higher type*, vol. 13. Springer Science & Business Media. (cited on page 11)

Pennington, J.; Socher, R.; and Manning, C., 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543. (cited on pages 20 and 25)

Pinterest, 2018. Mobile app design. https://www.pinterest.com.au/timoa/mobile-ui-photos/. Accessed 2018. (cited on pages 36 and 53)

Pttrns, 2018. Pttrns. https://pttrns.com/. Accessed 2018. (cited on pages 4 and 36)

Quinlan, J. R., 1983. Learning efficient classification procedures and their application to chess end games. In *Machine learning*, 463–482. Springer. (cited on page 26)

Radford, A.; Metz, L.; and Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, (2015). (cited on pages xv, 61, and 62)

Reiss, S. P.; Miao, Y.; and Xin, Q., 2018. Seeking the user interface. *Automated Software Engineering*, 25, 1 (2018), 157–193. (cited on pages 10 and 37)

Ren, S.; He, K.; Girshick, R.; and Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 91–99. (cited on pages xiii, 5, 37, and 42)

Ryan, C. and Gonsalves, A., 2005. The effect of context and application type on mobile usability: an empirical study. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, 115–124. Australian Computer Society, Inc. (cited on page 2)

Schwarz, M. W.; Cowan, W. B.; and Beatty, J. C., 1987. An experimental comparison of rgb, yiq, lab, hsv, and opponent color models. *ACM Transactions on Graphics (TOG)*, 6, 2 (1987), 123–158. (cited on page 45)

Shaik, K. B.; Ganesan, P.; Kalist, V.; Sathish, B.; and Jenitha, J. M. M., 2015. Comparative study of skin color detection and segmentation in hsv and ycbcr color space. *Procedia Computer Science*, 57 (2015), 41–48. (cited on page 45)

Shneiderman, B. and Plaisant, C., 2010. *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education India. (cited on page 2)

Silvennoinen, J.; Vogel, M.; and Kujala, S., 2014. Experiencing visual usability and aesthetics in two mobile application contexts. *Journal of usability studies*, 10, 1 (2014), 46–62. (cited on page 6)

Simonyan, K.; Vedaldi, A.; and Zisserman, A., 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, (2013). (cited on pages 17 and 21)

Singh, S. K.; Chauhan, D.; Vatsa, M.; and Singh, R., 2003. A robust skin color based face detection algorithm. *Tamkang journal of science and engineering*, 6, 4 (2003), 227–234. (cited on page 45)

Su, T.; Meng, G.; Chen, Y.; Wu, K.; Yang, W.; Yao, Y.; Pu, G.; Liu, Y.; and Su, Z., 2017. Guided, stochastic model-based gui testing of android apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 245–256. ACM. (cited on page 39)

Subramanya, S. and Yi, B. K., 2007. Enhancing the user experience in mobile phones. *IEEE Computer*, 40, 12 (2007), 114–117. (cited on page 6)

Swearngin, A.; Dontcheva, M.; Li, W.; Brandt, J.; Dixon, M.; and Ko, A. J., 2018. Rewire: Interface design assistance from examples. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 504. ACM. (cited on pages 10 and 37)

Szekely, P.; Luo, P.; and Neches, R., 1993. Beyond interface builders: Model-based interface tools. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*, 383–390. ACM. (cited on page 6)

Twitter, 2019. https://twitter.com. (cited on page 53)

Uijlings, J. R.; Van De Sande, K. E.; Gevers, T.; and Smeulders, A. W., 2013. Selective search for object recognition. *International journal of computer vision*, 104, 2 (2013), 154–171. (cited on pages 43 and 44)

Uplab, 2018. The global network for creatives get started. https://www.uplabs.com. Accessed: 2018-04-25. (cited on pages 4, 36, and 55)

Wang, C.; Yan, S.; Zhang, L.; and Zhang, H.-J., 2009. Multi-label sparse coding for automatic image annotation. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 1643–1650. IEEE. (cited on page 10)

Wang, H.; Ding, C.; and Huang, H., 2010a. Multi-label linear discriminant analysis. In *European Conference on Computer Vision*, 126–139. Springer. (cited on page 11)

Wang, M.; Ni, B.; Hua, X.-S.; and Chua, T.-S., 2012. Assistive tagging: A survey of multimedia tagging with human-computer joint exploration. *ACM Computing Surveys (CSUR)*, 44, 4 (2012), 25. (cited on page 11)

Wang, X.-Y.; Wu, J.-F.; and Yang, H.-Y., 2010b. Robust image retrieval based on color histogram of local feature regions. *Multimedia Tools and Applications*, 49, 2 (2010), 323–345. (cited on page 25)

Wang, Z. and Bovik, A. C., 2002. A universal image quality index. *IEEE signal processing letters*, 9, 3 (2002), 81–84. (cited on pages 43 and 44)

Wang, Z. and Bovik, A. C., 2009. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE signal processing magazine*, 26, 1 (2009), 98–117. (cited on page 43)

Wang, Z.; Bovik, A. C.; Sheikh, H. R.; Simoncelli, E. P.; et al., 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13, 4 (2004), 600–612. (cited on pages 43 and 44)

Winograd, T., 1995. From programming environments to environments for designing. *Communications of the ACM*, 38, 6 (1995), 65–74. (cited on page 2)

Yang, Y. and Klemmer, S. R., 2009. Aesthetics matter: leveraging design heuristics to synthesize visually satisfying handheld interfaces. In *CHI'09 Extended Abstracts on Human Factors in Computing Systems*, 4183–4188. ACM. (cited on page 6)

Yarkoni, T.; Balota, D.; and Yap, M., 2008. Moving beyond coltheartâĂŹs n: A new measure of orthographic similarity. *Psychonomic Bulletin & Review*, 15, 5 (2008), 971–979. (cited on page 42)

Yosinski, J.; Clune, J.; Bengio, Y.; and Lipson, H., 2014. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, 3320–3328. (cited on page 25)

Zen, M. and Vanderdonckt, J., 2014. Towards an evaluation of graphical user interfaces aesthetics based on metrics. In *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, 1–12. IEEE. (cited on page 6)

Zheng, S.; Hu, Z.; and Ma, Y., 2019. Faceoff: Assisting the manifestation design of web graphical user interface. (2019). (cited on pages 10 and 37)

Chapter 4

Appendix

.1 Appendix A: Independent Study Contract

Australian National University

INDEPENDENT STUDY CONTRACT HONOURS

Note: Enrolment is subject to approval by the projects co-ordinator

SECTION A (Students and Supervisors)

UniID: u6063820

SURNAME: Sidong FIRST NAMES: Feng

PROJECT SUPERVISOR (*may be external*): Dr Zhenchang Xing

COURSE SUPERVISOR (*a RSCS academic*): Dr Zhenchang Xing

COURSE CODE, TITLE AND UNIT: Comp4540, Software Engineering Research Project, 24 units

SEMESTER ☑ S1 ☐ S2 YEAR: 2019

PROJECT TITLE:

Design Composition Critique

LEARNING OBJECTIVES:

Show understanding of performance and critique of UI design
Collect, pre-process large-scale and high quality dataset
Identify the key aspects of human psychology and suggest designs
Effectively present research, methods and outcomes in oral, written, graphical forms

PROJECT DESCRIPTION:

Determine the look and feel of mobile components is an essential step of mobile application design. UI designers easily get stuck on finding an appropriate component to match with their design. The existing design kits either have only limited numbers of UI designs or lack effective ways of searching, filtering, and comparing the design resources.

The project is to collect large-scale and high quality UI designs of different types of UI components from screenshots of existing mobile apps. Analyse components and composition to provide inspirations for users.

Further, to build a system Mobile UI Gallery to apply a user-centred focus to the development and design of user interface.

Australian National University

ASSESSMENT (as per course's project rules web page, with the differences noted below):

Assessed project components:	**% of mark**	**Due date**	**Evaluated by:**
Thesis	85 (85%)		
Presentation	10 (10%)		
Critical Feedback	5 (5%)		

MEETING DATES (IF KNOWN):

STUDENT DECLARATION:
I agree to fulfil the above defined contract

…….. ……..
Signature Date

SECTION B (Supervisor):

I am willing to supervise and support this project. I have checked the student's academic record and believe this student can complete the project. I nominate the following reviewers and have obtained their consent to review the completed thesis (through signature or attached email)

…….. ……..
Signature Date

Reviewer 1:
Name: …….. Signature…….

Reviewer 2:
Name: …….. Signature…….

*Nominated reviewers may be subject to change on request by the supervisor.

REQUIRED DEPARTMENT RESOURCES:

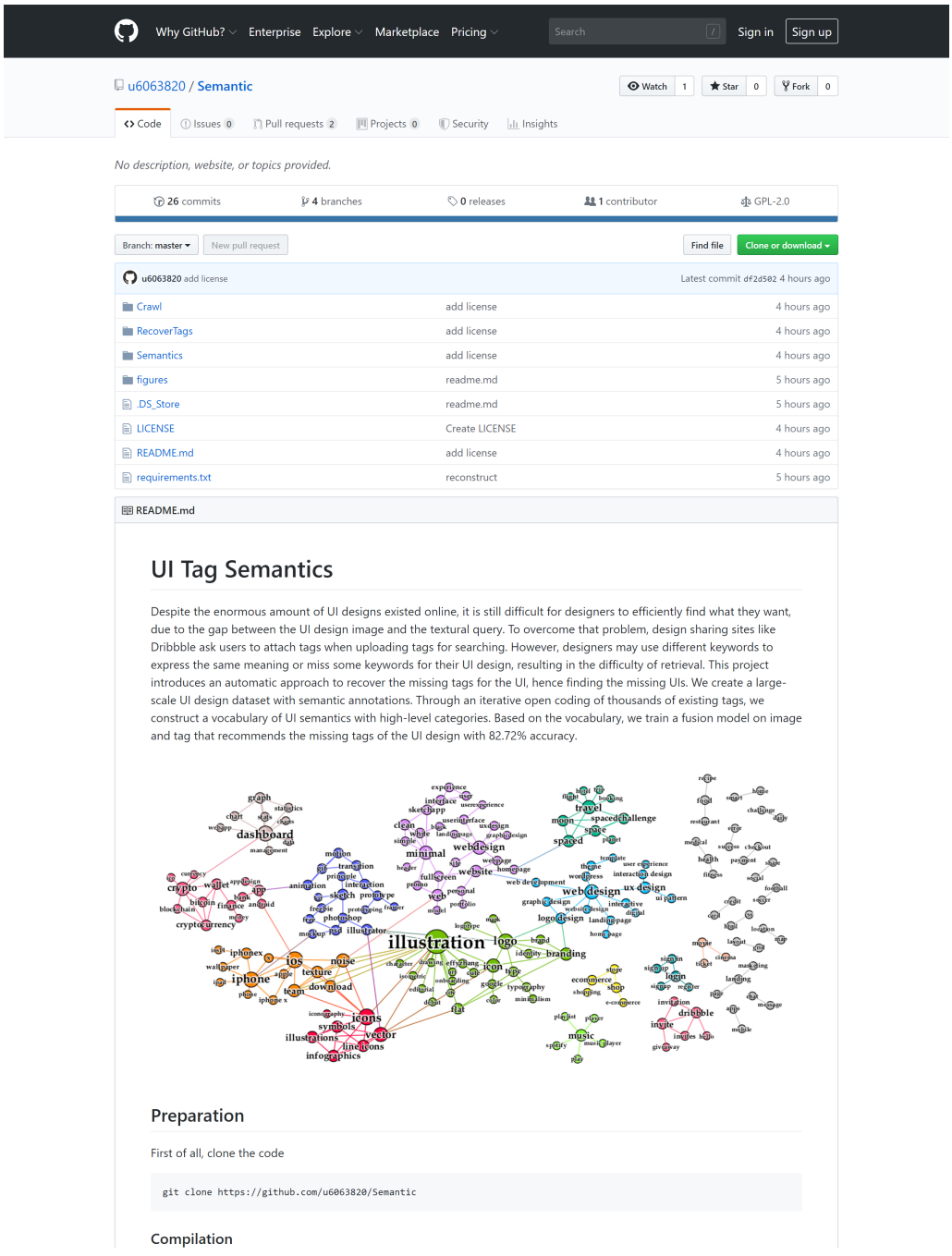
SECTION C (Course coordinator approval)

…….. ……..
Signature Date

SECTION D (Projects coordinator approval)

…….. ……..
Signature Date

.2 Appendix B: Artefacts for Whole UI Search



Install all the python dependencies using pip:

```
pip install -r requirements.txt
```

Usage

- Crawl designs from Dribbble

```
cd Crawl
```

Please follow the instructions in Crawl to prepare setups. Actually, you can refer to any others. After downloading the data, create softlinks in the folder ./Semantics/ and ./RecoverTags/Data/.

- Distill semantics of UI tags

```
cd Semantics
```

Please follow the instructions in Semantics.

- Predict missing tags

```
cd RecoverTags
```

Please follow the instructions in RecoverTags.

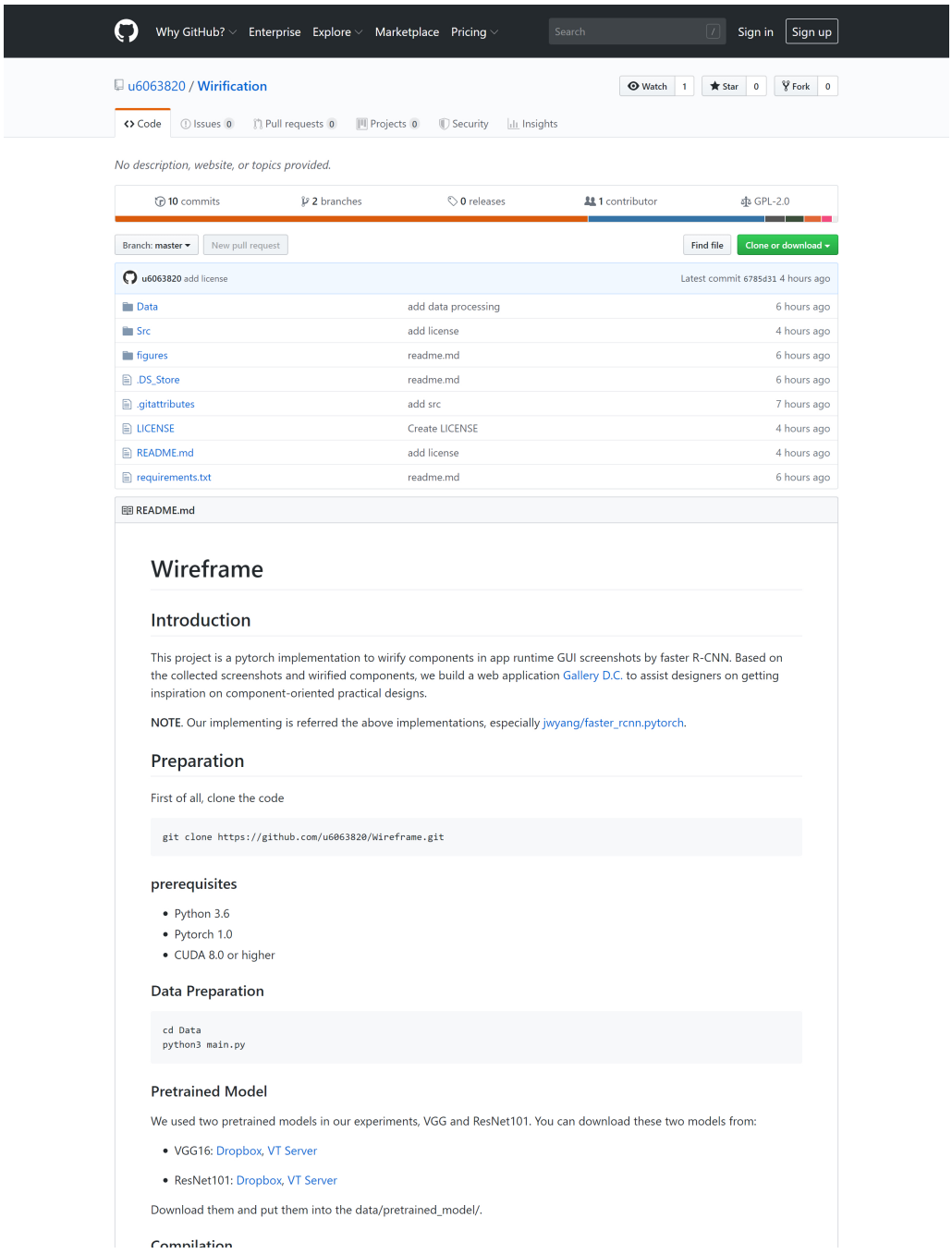
Authorship

This project is contributed by Sidong Feng.

License

License GPL v2

.3 Appendix C: Artefacts for Component-oriented Search



Why GitHub? Enterprise Explore Marketplace Pricing Search / Sign in Sign up

u6063820 / Wirification

Watch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Security Insights

No description, website, or topics provided.

10 commits 2 branches 0 releases 1 contributor GPL-2.0

Branch: master New pull request Find file Clone or download

u6063820 add license Latest commit 6785d31 4 hours ago

Data	add data processing	6 hours ago
Src	add license	4 hours ago
figures	readme.md	6 hours ago
.DS_Store	readme.md	6 hours ago
.gitattributes	add src	7 hours ago
LICENSE	Create LICENSE	4 hours ago
README.md	add license	4 hours ago
requirements.txt	readme.md	6 hours ago

README.md

Wireframe

Introduction

This project is a pytorch implementation to wirify components in app runtime GUI screenshots by faster R-CNN. Based on the collected screenshots and wirified components, we build a web application Gallery D.C. to assist designers on getting inspiration on component-oriented practical designs.

NOTE. Our implementing is referred the above implementations, especially jwyang/faster_rcnn.pytorch.

Preparation

First of all, clone the code

```
git clone https://github.com/u6063820/Wireframe.git
```

prerequisites

- Python 3.6
- Pytorch 1.0
- CUDA 8.0 or higher

Data Preparation

```
cd Data
python3 main.py
```

Pretrained Model

We used two pretrained models in our experiments, VGG and ResNet101. You can download these two models from:

- VGG16: Dropbox, VT Server
- ResNet101: Dropbox, VT Server

Download them and put them into the data/pretrained_model/.

Compilation

Compilation

Install all the python dependencies using pip:

```
pip install -r requirements.txt
```

Compile the cuda dependencies using following simple commands:

```
cd Src/lib
python3 setup.py build develop
```

It will compile all the modules you need, including NMS, ROI_Pooing, ROI_Align and ROI_Crop.

Train

To train a faster R-CNN model with vgg16 on dataset, simply run:

```
CUDA_VISIBLE_DEVICES=$GPU_ID python trainval_net.py \
                   --dataset pascal_voc --net vgg16 \
                   --bs $BATCH_SIZE --nw $WORKER_NUMBER \
                   --lr $LEARNING_RATE --lr_decay_step $DECAY_STEP \
                   --cuda
```

where 'bs' is the batch size with default 1.

Test

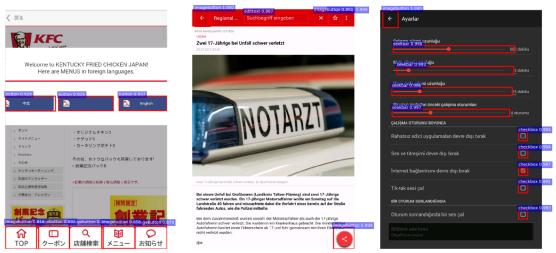
If you want to evlauate the detection performance of a pre-trained vgg16 model on test set, simply run

```
python test_net.py --dataset pascal_voc --net vgg16 \
                   --checksession $SESSION --checkepoch $EPOCH --checkpoint $CHECKPOINT \
                   --cuda
```

Specify the specific model session, chechepoch and checkpoint, e.g., SESSION=1, EPOCH=6, CHECKPOINT=100.

Detection Results

Below are some detection results:



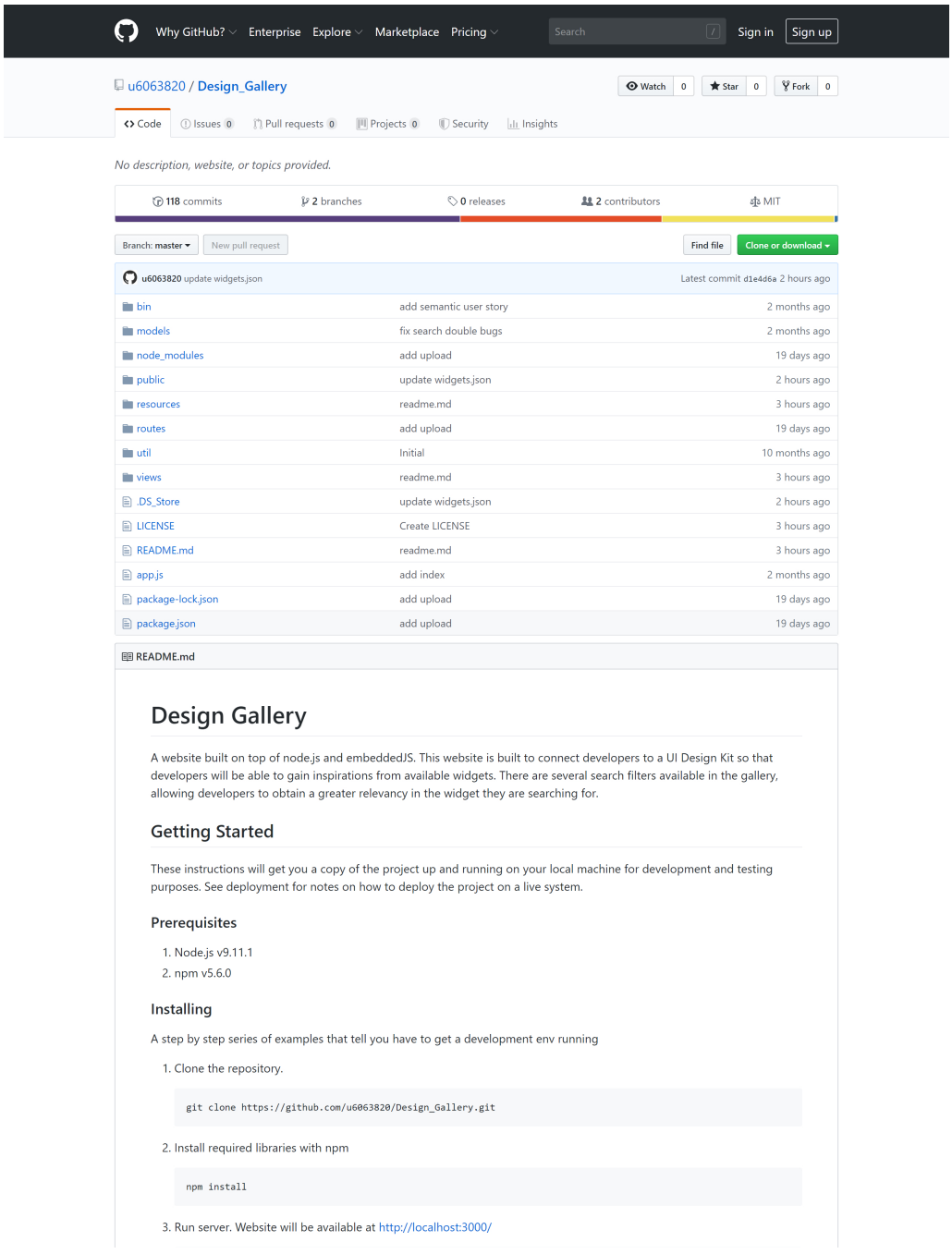
Authorship

This project is contributed by Sidong Feng.

License

License GPL v2

.4 Appendix D: Artefacts for Design Gallery



```
npm start
```

Note. It is recommended to use nodemon as changes will be made dynamically.

```
```
nodemon npm start
```
```

Deployment

Deployment is done via Google Cloud Storage and Heroku. Due to the fact that Google does not support node.js deployment using App Engine (requires the use of Flexible Environment that might incur huge charges), we decided to implement the app on Heroku instead which is free.

Obtain a heroku account and deploy the entire app onto heroku by following instructions on Heroku itself.

There is also a requirement for a database to be setup. We use the free MongoDB host. You should be able to get a MongoDB API link which will be added into Heroku under `Settings > Config Vars` with the values `MLAB_API_LINK=mongodb://USER:PW@XXX.mlab.com`.

The data stored in MongoDB is within the `widgets` collection with the following item structure:


```
{
    "_id" : ObjectId("5ac23b7e27bc4099fb9fb172"),
    "id":0,
    "name":"Button-0",
    "widget_class":"CheckBox",
    "color":{
        "Blue":0.0,
        "Yellow":0.0,
        "Green":0.0,
        "Cyan":0.0,
        "Black":1.0,
        "White":0.0,
        "Magenta":0.0,
        "Red":0.0,
        "Lime":0.0
    },
    "coordinates":{
        "to":[728,220],
        "from":[685,177]
    },

    "dimensions":{
        "width":43,
        "height":43
    },
    "text":"Hello",
    "font":"Bosis Pro Light",
    "sims":["CheckBox-66844","CheckBox-27801","CheckBox-40382"],
    "screenshot":"a2dp.Vol-0.png",
    "Developer":"JimRoal",
    "url":"https://play.google.com/store/apps/details?id=a2dp.Vol",
    "application_name":"A2DP Volume",
    "package_name":"a2dp.Vol",
    "category":"TRANSPORTATION",
    "date":"2015-01-01",
    "downloads":"100,000 - 500,000"
}
```


Widgets are stored on Google Cloud Storage with the following directory structure.

```
.
├── widgets
└── screenshot
```

Authorship

This project is contributed by Sidong Feng.

License

License MIT

.5 Appendix E: Manual Script for Whole UI Search

Tool objective:

- **Correctly recovering missing tags.** For example, one designer uploads a sport design but forgets to write sport tag / misspells with sporrt. Dribbble is unable to display this design when searching sport. That would be a great loss for designers and uploaders.
- **Help designers to find more relevant results.** For example, designer looks for ios trip list ui designs. In Dribbble, they return 14 images: too little to find inspirations, thus designer needs to reduce keywords to mobile travel (normally large number of images but most not relevant to list) and manually finds list ui design (time consuming). In contrast, our gallery returns more results: 180 images, provides more inspirations to designer.
- **Rank the retrieval results to show the most relevant ones.** For example, if designers search ios trip list ui, there will be 180 images. Most of designers care more about the most relevant and will not view all of them. Therefore, ranked results based on their relevance to search terms are shown and it can save time for designers.

Tool Usage Instruction:

1. Search **keywords** in query box (split in space). For example, *Iphone Pay*
2. (Optional) Tick **categories** (i.e., platform, color) to see ranked results in category split (top 32 images are displayed).
3. Click **Search** button to see the result.
4. Click **link to dribbble** to see the corresponding result in Dribbble.
5. (Optional) Click **Link to local dribbble result** to see the retrieval results by matching the keywords with the same tags.

Organize Search Results by the Selected Categories:

platform	mobile, website
color	grey, brown, pink, black, white, green, blue, red, yellow
app	weather, social, sport, news, health, game, finance, travel, food, ecommerce, music
function	profile, signup, checkout, landing
layout	chart, grid, form, list, dashboard

Example Search query:

- *Iphone Blue Checkout*
- *Ios Trip List*
- *Black Finance Chart*
- *Website Music List*

Note:

- We crawl 16549 images from Dribbble as our database. Some queries may have fewer results than Dribbble.