Deep Learning UI Design Patterns of Mobile Apps

Tam The Nguyen[1], Phong Minh Vu[1], Hung Viet Pham[2], Tung Thanh Nguyen[1]
[1]Auburn University, [2]Utah State University
tam@auburn.edu,lenniel@auburn.edu,hv.pham.2704@gmail.com,tung@auburn.edu

ABSTRACT

User interface (UI) is one of the most important components of a mobile app and strongly influences users' perception of the app. However, UI design tasks are typically manual and time-consuming. This paper proposes a novel approach to (semi)-automate those tasks. Our key idea is to develop and deploy advanced deep learning models based on recurrent neural networks (RNN) and generative adversarial networks (GAN) to learn UI design patterns from millions of currently available mobile apps. Once trained, those models can be used to search for UI design samples given user-provided descriptions written in natural language and generate professional-looking UI designs from simpler, less elegant design drafts.

1 INTRODUCTION

User interface (UI) is one of the most important components of a mobile app because users mainly interact with the app via UI (e.g., tap/click on an icon or scroll/swipe a page). Thus, UI strongly influences users' perception of and experience with the app, which could decide whether the app is successful or not.

To design beautiful, appealing apps, app development teams often hire well-trained UI/UX designers. However, even with high-skilled UI designers, the UI design tasks are mostly manual, and thus, very time-consuming. To address this problem, this paper proposes DeepUI, a novel approach to (semi)-automate those tasks based on learning UI design patterns.

A UI design pattern is a design idea or template that is used in different apps (or in different places in the same app). For example, most apps have a login screen with UI elements for user id (e.g., username, email, or phone number), password, and sometimes third-party authentication methods (e.g., login via Facebook, Google, or Twitter accounts).

The key idea of DeepUI is to develop and deploy advanced deep learning models based on recurrent neural network (RNN) and generative adversarial network (GAN) to learn UI design patterns from millions of mobile apps currently available on app stores. After learning, those models can be used to support app developers in designing UI of new apps with the UI design patterns they learned. Let us illustrate our approach via two use cases.

1) *Search for UI design samples via user-provided descriptions written in natural language.* Assume a scenario in which a developer needs to design a login screen for a new mobile app. There are several requirements for the screen, such as users will enter email and password to login, the screen will include the logo of the app and have one or multiple alternate login methods, etc. At the same time, the developer does not want to reinvent the wheel by designing the user interface of the login screen from ground up. He wants to create a login screen that meets the requirement and follows a popular UI design pattern of existing apps.

The app developer writes the description of the screen he wants in a natural language like English. Then, he uses that description as a query to search against DeepUI's repository of millions of UI design samples and templates. DeepUI matches and shows design samples having descriptions similar to what the developer describes. He can further filter and rank the search results with other criteria like apps' ratings or categories.

Collecting millions of UI design samples can be done simply by downloading millions of apps available on app stores, running them (e.g., automated via DroidMate[5]), and scraping their UI screens. However, it is impractical to manually describe those samples in natural language. Inspired by *image captioning* techniques based on deep learning models, we propose NaturalUI, a deep learning model for learning natural language descriptions of UI designs. Once trained, NaturalUI can generate natural language descriptions for UI designs collected by DeepUI, thus, enabling the search of UI design samples using descriptions written in natural languages.

2) *Generate professional-looking UI designs from simpler, less elegant design drafts.* Assume a scenario in which a developer needs to design a user screen for an app. To do that, the developer has come up with a prototype design which includes several wireframes. Each wireframe depicts the screen layout or arrangement of the content, including interface elements and navigational systems, and how they work together. As an initial design of the screen, a wireframe is often incomplete. It is also too generic because in a wireframe the developer often focuses only on the layout of elements. The developer also needs to design the user interface for the screen from scratch based on the design in the wireframe. This task could be frustrated for the developer especially if he does not have prior experiences in user interface design or front-end programming.

DeepUI helps in this situation by generating elegant, professional-looking UI design from the prototype design (wireframes) for the developer. Its core model is GenUI, a deep learning model based on Generative Adversarial Networks (GAN) [4]. As a GAN, GenUI has two main components: UIGenerator and UIDiscriminator. Both components are trained via a repository of elegant, professional-looking UI designs collected from millions of existing mobile apps. UIGenerator is trained to generate new ("fake" in GAN terminology) designs that look like the existing ("true" in GAN terminology) ones, while UIDiscriminator aims to discriminate the "fake" and the "true" ones. once trained, UIGenerator can be able to generate the "fake" that look highly similar to the "true" ones.

ICSE-NIER'18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5662-6/18/05…$15.00
https://doi.org/10.1145/3183399.3183422

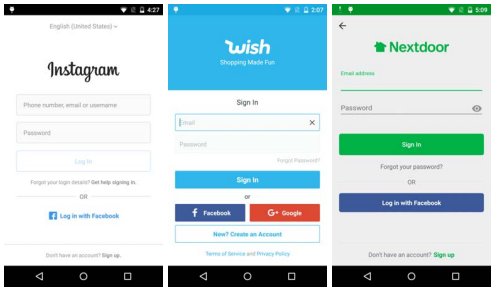

Figure 1: **Login screens for Instagram, Wish, and Nextdoor**

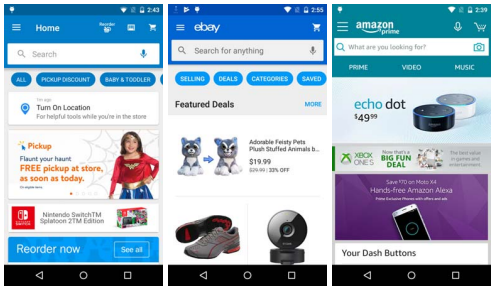

Figure 2: **Main screens of Walmart, Ebay, and Amazon**

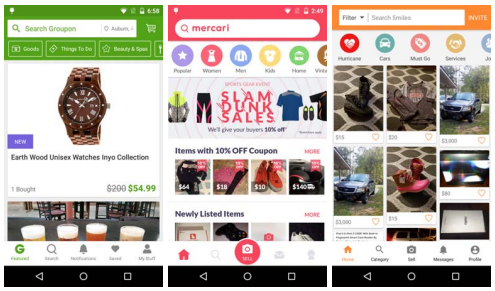

Figure 3: **Main screens of Groupon, Mercari, and 5miles**

Based on the provided prototype design and its description, UIGenerator will generate new UI designs having the same structure and description with the input, but the appearance is much more like ones in the repository.

In the remaining of this paper, we will discuss our preliminary observations of mobile apps' UI design patterns in Section 2. In Section 3, we describe our two models as the core of DeepUI. Section 4 discusses related work and conclusions.

2 MOTIVATION EXAMPLES

In this section, we describe our preliminary result on studying user interface (UI) design patterns for mobile apps. Different apps often have similar feature or functionality, thus, could have similar UI design patterns for screens associated with those features or functionality. In the first example, we choose to study login screens, a very popular screen that most apps include. We collected login

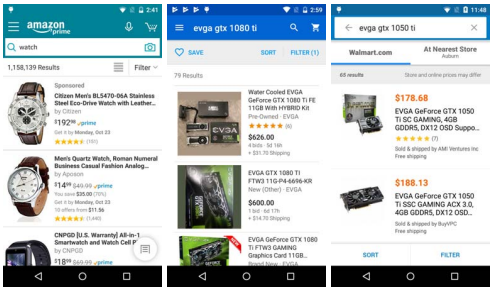

Figure 4: **Search screens of Walmart, Ebay, and Amazon**

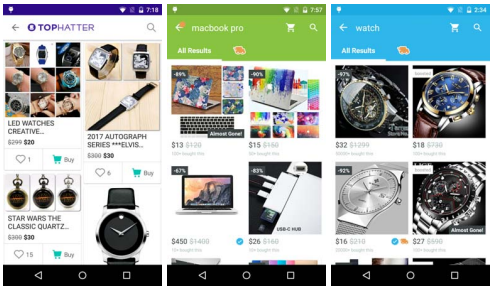

Figure 5: **Search screens of Tophatter, Geek, and Wish**

screens of 40 top-free Android apps from 4 categories: Communication, Education, Shopping, and Social in the Google Play Store. The extraction process is described as follows. Each app was downloaded and installed in a Moto G (1st generation) connected to a computer. We then used the tool UI Automator Viewer (integrated in the Android SDK) to extract the screens and the corresponding view layout hierarchies. After that we manually inspected those screens to observe UI design patterns exhibit among them.

We observed that there are mainly three patterns for login screens, namely: login with phone number, login with username and password, and login with only email. Next, let us describe a pattern in detail (note that we describe UI elements in top-bottom, left-right order). Figure 1 shows three login with username and password screens. From the figure, we could see that the login screens in the three apps not only have similar UI elements in functionality but also shares a similar design layout. All three screens have an image view to display app logo in the top of the screen. Next, they have two text boxes for entering username and password. Both text boxes include placeholders to guide users to enter correct information. The pattern also has a clickable "Forgot password" text label, and a sign in/login button. This design also includes sign in button(s) using information from other app, such as, Facebook and Google. At the bottom of screens, there is a clickable text label or a button for signing up if users are new to the app and do not have accounts before.

In the next case study, we examined UI design patterns in the main screen of shopping apps. To do that we collected main screens of 30 top-free shopping apps from the Google Play Store. After manually inspecting the screens we identified that there are two general UI design patterns exhibiting among those screens. Let us

describe the patterns in detail. Figure 2 shows three main screens of Walmart, Ebay, and Amazon representing the first pattern. In general, these screens consist four parts: a menu bar, a search bar, a navigation bar, and a content layout. As we can see from the figure, a menu bar often consists an image button for opening the menu, a logo or title, and a button for showing the shopping cart. A menu bar can also include other buttons, e.g. a reorder button and a barcode button in Walmart, a voice button in Amazon. A search bar often consists text box for users to enter a search query. It also has a image search button on the left and a voice or image search button on the right. A navigation bar often has a list of buttons. Each button is linked to a part or feature of the app. Finally, these apps have a content layout which occupies the majority of the screen. Note that UI elements in the content layout are different betweens apps as each app has an own way to present contents.

The second design pattern is depicted via the main screens of three apps Groupon, Mercari, and 5miles showed in the Figure 3. We can see that the design of screens is different from the previous design. The search bar is placed in the top of the screen. The names of the apps are integrated into the place holder of the search box. Below the search bar is the category bar. It contains a list of category buttons. Each category button contains an image and a text represents the name of corresponding category. Interestingly, Mercari and 5miles have quite similar design of category buttons compared to Groupon. The navigation bar is placed in the bottom of the screens with buttons for features such as Home page, search, sell, and account.

In the final case study, we compared the search result screens of shopping apps. We also found two patterns depicted in Figures 4 and 5. The general structure of the first pattern includes a navigation and/or a search bar in top of the screen, and a vertical list of result entries. The navigation often includes a menu button, a text label, and a shopping cart button (in Amazon and Ebay apps). All the screens include a text label showing the number of results found for a search, a button for sorting result entries, and a button filtering result entries. Although, the location of those button might be different between apps, e.g. the sort and filter buttons in Amazon and Ebay are placed above the result list while for Walmart, they are place in the bottom of the screen. The design the a result entry in three apps is also similar. Each entry has an image showing the item in the left. In the right, it shows information about the item such as the title, the manufacturer, condition, price, and shipping.

Figure 5 shows the search result screens for Tophatter, Geek, and Wish apps. We can see that these screens follow a design different from the previous design described above. While it still has a navigation bar in the top of the screen, the result list now is a grid of result entries. Each result entry is a rectangle cell in the grid. The design of a result entry is also different from the previous design. The image showing the item occupies the majority portion of a result entry, while other information such as price, the number of users liked or bought the item are placed below the image.

3 APPROACH

3.1 Generating natural descriptions

The input of NaturalUI is a dataset of UI instances and their descriptions. An UI instance could be a screen, a part of screen, or

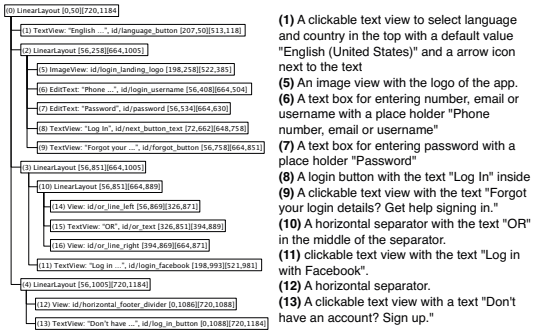

Figure 6: **A sample data in the training dataset of NaturalUI**

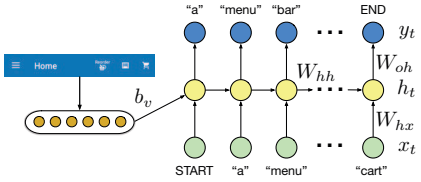

Figure 7: **The structure of NaturalUI**

Figure 8: **Recommending UI design from a description**

an UI element and its description. Figure 6 shows an example of a data sample in the dataset corresponds with the login screen of Instagram showed in Figure 1. The dataset is constructed by manually labeling UI screens. Each UI instance includes a screenshot or image of UI elements. It also contains a view hierarchy tree that stores hierarchical structure of UI elements. Each node in the tree corresponds with an UI element and it stores information about the UI element such as, the coordinates of the element in the screens, its attributes, resources that the element points to, etc. Each UI element of the screen has a corresponding description.

The design of NaturalUI is inspired by Deep-Visual [6], a Recurrent Neural Network (RNN) [8] for generating natural language description for images. RNNs is a class of models that learn the probability distribution of the next word given the current and the previous context. In NaturalUI, we extend RNNs to also include the visual representation and structure of UI elements as an input for the model. More formally, we design a RNN that takes an UI instance and a sequence of words $(x_1, x_2, ..., x_t)$ in the description as input. It then computes a sequence of hidden states $(h_1, h_2, ..., h_t)$ and a sequence of outputs $(y_1, ..., y_t)$.

Figure 7 shows the the overall structure of NaturalUI. The training process of NaturalUI is proceeded as follows. x_1 is set to a special *START* vector indicates the start of a sentence, the context vector h_1 is initialized by combining x_1 with the semantic information of

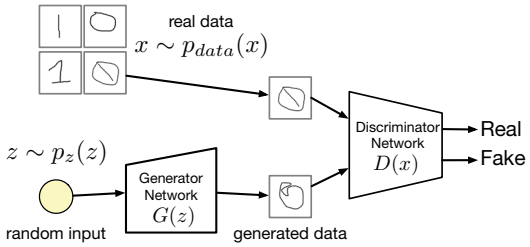

Figure 9: **Generative Adversarial Networks (GANs)**

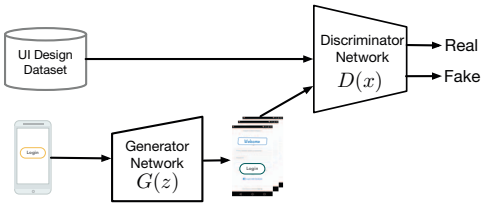

Figure 10: **The structure of GenUI**

the UI instance (b_v). b_v could be learned by training a Recursive Neural Network [9]. The desired label y_1 is set to the first word of the sentence. Similarly, we set x_2 as the word vector of the first word and expect the network predict the second word. Finally, in the last step, x_t represents last word, the target label is set to a special *END* token indicate the end of a sentence. The cost function to maximize is the log probability assigned to target labels.

To generate a description for an UI instance, NaturalUI first computes the semantic representation of the instance b_v, x_1 is set to the *START* vector. From that it updates the context h_1 and computes the distribution over the first word y_1. A first word is sample from the distribution by picking the argmax. x_2 is set as the word vector of first word, and NaturalUI repeats the process until the *END* token is generated.

Based on NaturalUI, a system for recommending UI design from a description is depict in Figure 8. Given a natural language description of a design, the system finds top similar UI design descriptions generated from NaturalUI. The system then recommends the UI design instance corresponding with those descriptions.

3.2 Generating user interface design

GAN [4] is a framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G and a discriminative model D. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. The structure of the networks is showed in Figure 9.

An random input z is sampled from a probability distribution $p_z(z)$. The generator network $G(z)$ then takes the random input z and tries to generate a sample of data. The generated samples is then fed into the discriminative network $D(x)$. The task of the discriminative network $D(x)$ is to take input either from the real data (x is sampled from the probability distribution $p_{data}(x)$) or from the generator and try to predict whether the input is real or generated. $D(x)$ solves a binary classification problem using sigmoid function giving output in the range of 0 to 1.

GenUI is proposed based on the architecture of GAN. Figure 10 shows the overall structure of the networks. The training data is a repository of elegant, professional-looking UI designs collected from millions of existing mobile apps. Each instance includes the screen itself, the view hierarchy structure, and all resources associated with its UI elements. Given a prototype design instance z, the generator will generate a set of UI design instances $G(z)$. $G(z)$ is then fed into the discriminative network $D(x)$. The task of the discriminative network $D(x)$ is to take input either from the real data or from the generator and try to predict whether the input is "real" or "fake". The model is trained until the discriminator cannot distinguish the real data and the generated data from the generator. The generator then can be used to generate new UI designs having the same structure and description with the input, but the appearance is much more like ones in the repository.

4 RELATED WORK

So far as we know of, there has been no previous work for mining design patterns from mobile app UI. Our work was inspired by ERICA [3], a system designed by Deka et al. for collecting GUI interactions from mobile apps. However, in ERICA, the goal and techniques used were focusing on mining the interaction between user and app, while our goal is to mine design patterns, hence the differences in our approach as well. In his follow up works, Deka presented an approach to app design mining from the backbone of ERICA [1, 2]. These works provided a dataset for learning app designs from 9.7k apps. Although, no known research has been using this dataset for learning the designs of apps.

On the aspect of design patterns, Talton et al. proposed a learning method using Bayesian Grammar Induction [10] and is applied on web designs, architecture designs, etc. Nonetheless, this method has not been applied on mobile app design, and yet to explore the characteristic and domain differences of it. Similar to that work, Webzeitgeist [7] by Kumar et al. was focusing on mining the design of websites. Despite of having similar analogy, such as both app designs and web designs use a markup language for their structure, they are vastly different in term of appearance, usages and platforms. Therefore, our study does not cross path with them.

REFERENCES

[1] B. Deka. 2016. Data-driven Mobile App Design. In *UIST*.
[2] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. (2017).
[3] B. Deka, Z. Huang, and R. Kumar. 2016. ERICA: Interaction Mining Mobile Apps. In *UIST*.
[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. Generative Adversarial Nets. In *NIPS*.
[5] K. Jamrozik and A. Zeller. 2016. DroidMate: A Robust and Extensible Test Generator for Android *(MOBILESoft '16)*.
[6] A. Karpathy and L. Fei-Fei. 2017. Deep Visual-Semantic Alignments for Generating Image Descriptions. *IEEE Trans. Pattern Anal. Mach. Intell.* (2017).
[7] R. Kumar, A. Satyanarayan, C. Torres, M. Lim, S. Ahmad, S. R Klemmer, and J. Talton. 2013. Webzeitgeist: design mining the web. In *SIGCHI*.
[8] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*.
[9] R. Socher, C. Lin, A. Y. Ng, and C. Manning. 2011. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *ICML*.
[10] J. Talton, L. Yang, R. Kumar, M. Lim, N. Goodman, and R. Měch. 2012. Learning design patterns with bayesian grammar induction. In *UIST*.