Auto-Icon: An Automated Code Generation Tool for Icon Designs Assisting in UI Development

Sidong Feng
Suyu Ma
Faculty of Information Technology
Monash University
Melbourne, Australia
sidong.feng@monash.edu,suyu.ma1@monash.edu

Jinzhong Yu
Alibaba Group
Hangzhou, China
jinzhong.yjz@alibaba-inc.com

Chunyang Chen
Faculty of Information Technology
Monash University
Melbourne, Australia
chunyang.chen@monash.edu

Tingting Zhou
Yankun Zhen
Alibaba Group
Hangzhou, China
miaojing@taobao.com,shadow2597758@gmail.com

ABSTRACT

Approximately 50% of development resources are devoted to UI development tasks [8]. Occupied a large proportion of development resources, developing icons can be a time-consuming task, because developers need to consider not only effective implementation methods but also easy-to-understand descriptions. In this study, we define 100 icon classes through an iterative open coding for the existing icon design sharing website. Based on a deep learning model and computer vision methods, we propose an approach to automatically convert icon images to fonts with descriptive labels, thereby reducing the laborious manual effort for developers and facilitating UI development. We quantitatively evaluate the quality of our method in the real world UI development environment and demonstrate that our method offers developers accurate, efficient, readable, and usable code for icon images, in terms of saving 65.2% developing time.

CCS CONCEPTS

• **Human-centered computing → Empirical studies in accessibility**.

KEYWORDS

code accessibility, icon design, neural networks

ACM Reference Format:

Sidong Feng, Suyu Ma, Jinzhong Yu, Chunyang Chen, Tingting Zhou, and Yankun Zhen. 2021. Auto-Icon: An Automated Code Generation Tool for Icon Designs Assisting in UI Development. In *26th International Conference on Intelligent User Interfaces (IUI '21), April 14–17, 2021, College Station, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3397481.3450671

IUI '21, April 14–17, 2021, College Station, TX, USA

ACM ISBN 978-1-4503-8017-1/21/04...$15.00
https://doi.org/10.1145/3397481.3450671

1 INTRODUCTION

An user interface (UI) consists of series of elements, such as text, colors, images, widgets, etc. Designers are constantly focusing on icons as they are highly functional in an user interface [9, 39, 41, 51]. One of the biggest benefits of icons is that they can be universal. For instance, by adding a red "X" icon to your user interface design, users are informed that clicking this icon leads to the closure of a component. Furthermore, icons can make UIs look more engaging. For example, instead of using basic bullets or drop-downs filled with words, a themed group of icons can capture instant attention from users. Consequently, icons become an elegant yet efficient way to communicate with and help guide user through experience.

Despite of all these benefits, icons have two fundamental limitations in the day-to-day development environment, in terms of *rendering speed* and *code accessibility*. First, to ensure a smooth user interaction, UI should be rendered in under 16ms [22, 23, 30], while icon implemented as an image faces the slow rendering problem, due to image download speed, image loading efficiency, etc. These issues will directly affect the quality of the product and user experience, requiring more effort from developers to develop an advanced method to overcome the problem. Second, in the process of UI implementation, many developers directly import the icon image resources from the UI draft files without considering the meaning of the content, resulting in poor description/comment during coding. Different codes render the same visual effect to users, while it is different for developers to develop and maintain. A non-descriptive code increases the complexity and effort required to developers as they need to look at the associated location of the UI to understand the meaning of the code.

This challenge motivates us to develop a proactive tool to address the existing UI development limitations and improve the efficiency and accessibility of code. Our tool, *Auto-Icon*, involves three main features. First, to meet the requirement of efficient rendering, we develop an automated technique to convert icon image to icon font, which is a typeface font. Once the font is loaded, the icon will be rendered immediately without downloading the image resources, thereby reducing HTTP requests and improving the rendering speed. Icon font can further optimize the performance of rendering by adopting HTML5 offline storage. Besides, icon font has other potential attributes that can facilitate UI development, such as easy to use (i.e., use the CSS's @fontface attribute to load the font), flexible (i.e., capable to change color, lossless scale), etc. Second, understanding the meaning of icons is a challenging problem. There are numerous types of icons in the UIs. Icons representing the same meaning can have different styles and can be presented in different scales as shown in Table 1. Also, icons are often not co-located with texts explaining their meaning, making it difficult to understand from the context. In order to offer an easy access for developers to develop through understanding the meaning of icon, we collect 100k icons from existing icon sharing website Alibaba Iconfont [2] - each associating with a label described by designer. By analyzing the icon images and labels, we construct 100 categories, such as "left", "pay", "calendar", "house", etc. We then train a deep learning classification model to predict the category of the icon as its description. The experiments demonstrate that our model with the average accuracy as 0.87 in an efficient classification speed as 17.48ms, outperforms the other deep learning based models and computer vision based methods. Third, to provide more accessibility to developers on the description of icon images, we also detect the primary color of icons by adopting HSV color space [66]. We refer to our mechanism tool *Auto-Icon* to build an intelligent support for developers in the real context of UI development, assisting developing standardized and efficient code.

To demonstrate the usefulness of *Auto-Icon*, we carry out an user study to show if our tool for automatically converting an icon image to an icon font with label descriptions can help provide more knowledge on code accessibility and accelerate UI development for developers. After analyzing ten professional developers' feedback with all positive responses on our mechanism tool and we find that the code for icon image generated by our tool can achieve better readability compared with the code manually written by professional developers. Besides, *Auto-Icon* has been implemented and deployed in Alibaba *Imgcook* platform. The results demonstrates that our tool provides 84% usable code for icon images in a realistic development situations. Our contributions can be summarized below:

- We identify the fundamental limitations of existing UI development of icon images. The informal interviews with professional developers also confirm these issues qualitatively.
- Based on the emerging label categories, we develop deep-learning and computer-vision based techniques, called *Auto-Icon*, for specifically converting icon image to icon font with label describing its meaning and color to provide developers understand knowledge of code.
- We conduct large-scale experiments to evaluate the performance of our tool *Auto-Icon* and shows that our tool achieves good accuracy compared with baselines. The evaluation conducted with developers and tested on the *Imgcook* platform demonstrates the usefulness of our tool.
- We contribute to the IUI community by offering intelligent support for developers to efficiently develop icon images comply with code standardization.

2 RELATED WORKS

2.1 UI Rendering

Ensuring fast rendering speed is an essential part in UI development, since slow rendering creates poor user experience. Many studies focus on improving rendering speed via reducing bugs [11, 36, 45, 47, 49, 57, 60, 70]. In contrast, we focus on analyzing image displaying performance in UI rendering. There are a few related works in this domain. For example, Systrace [4] is a tool that allows developers to collect precise timing information about UI rendering on devices. However, it does not provide any suggestions for improvement. To address this problem, many studies introduce reliable approaches to improve rendering efficiency such as image resizing based on pattern-based analysis [48], a manual image resource management based on resource leakage analysis [72]. Gao et al. [29] implement a system called DRAW which aims to reveal UI performance problems in an application such as excessive overdraw and slow image components detection. With the suggestion of the image displaying performance analysis by DRAW, developers can manually improve the rendering performance of slow image displaying. While these works offer image management suggestions to developers to achieve better rendering performance, they still need to be improved manually. In contrast, we propose an image conversion technology based on computer vision and graphic algorithms to convert icon images into font types in order to automatically improve the performance of UI rendering.

2.2 Code Accessibility

Digital devices such as computer, mobile phone and tablets are widely used. To ensure the quality of software, many research works have been conducted [10, 28, 34]. Most of these works focus on the functionality and usability of apps such as GUI design [13, 15, 16, 74], GUI animation linting [78], localization [68], privacy and security [17, 18, 20, 25, 76], performance [47, 79], and energy-efficiency [6, 7]. Few research works are related to accessibility issues. Some works in Human-Computer Interaction area have explored the accessibility issues of mobile apps [14, 40, 54, 67, 75]. In these work, the lack of description in image-based components in UI is commonly regarded as an important accessibility issue. For instance, Harrison et al. [32] establish an initial 'kineticon vocabulary' containing a set of 39 kinetic behaviors for icon images, such as spin, bounce, running, etc. Ross et al. [61] identify some common labeling issues in Android apps via analyzing the icon image labeling. With crowd source method, Zhang et al [77] annotate GUI elements without content description. However, these works are mainly based on the support from developers. Due to the increasingly developed Convolutional Neural Networks (CNNs) technologies, dramatic advances appears in the field of image classification which is applied to automatically annotate tags for images. Chen et al. [12] analyze the tags associated with the whole GUI artwork collected from Dribbble, and emerge an vocabulary that summarizes the relationship between the tags. Based on the vocabulary, they adopt a classification model to recommend the general tags in the GUI, such as "sport", "food", etc. Different from their work, we predict more fine-grained categories, such as "football", "burger", etc. And also, they focus on predicting the categories of the whole UI which is subjective to human perception, but the categories of small icon images are usually more intuitive. A similar work to ours is the icon sensitive classification by Xiao et al [73]. They utilize traditional computer vision techniques like SIFT and FAST to extract the features of icon and classify icons into 8 categories through calculating their similarity. After the systematically investigation of icon images, we discover the fundamental limitations in icon images discussed in Section 4.1, in terms of *high cross-class similarity* and *small, transparent and low contrast*. These findings conflict with methods applied in their paper such as apply rotation to augment dataset. Moreover, we show that deep learning model is fruitful for the icon classification problem than the tradition computer vision technique in Section 5.1.3. In our work, according to the characteristic of icon images, we propose a deep learning model to automatically classify icons in a more fine-grained (100) category and also adopt a computer vision technique to detect its primary color.

3 PRELIMINARY STUDY

To better understand the challenges in the real-world UI development environment, we conducted an interview with 12 front-end developers from the big companies. Two authors first developed the interview protocol, and conducted pilot studies with two participants. Based on the pilot studies, we refined the interview protocol and conducted 10 interviews formally. The average length of these interviews is 20 minutes. We started with general questions, such as questions about working years, workload of development, and number of projects developed. Then, we asked the interviewees how they developed the code for icon images. We particularly asked what motivated them to adopt the approach, whether they revised the implementation, what approaches could achieve the same effect, what they perceived as the impact of the implementation, how the implementation behaved in the process of development and is there any difference on UI development between personal projects and company tasks.

3.1 Research Question 1: do developers implement icon images in font or images?

By summarising the approaches, we collected 4 ways of rendering icon images, i.e., image tag <img> or <svg>, icon tag <i>, css background image, and custom tag <SvgIcon> as shown in Table 3. One third of our developers listed all approaches, 80% developers knew the way of using image and font. There are 2 developers who have never heard of or used the fonts to render icon images, D2 said:

Making front-end development is fun, although sometimes it hurts because I do not have adequate learning experience. There are few front-end courses in universities, and these courses usually contain relatively simple knowledge, such as what is <div> block, how to connect HTML and CSS together, etc. They do not teach the usage of font, especially they do not distinguish the difference between fonts and images in rendering icons.

70% developers implemented icons as image when developing front-end codes based on UI design draft files because they found that converting icon to font is a complicated and laborious process. For example, D7 mentioned:

To implement the approach of icon font, I first need to upload the image to the existing conversion websites such as icomoon [38] and Fontello [26]. Then, I need to download the generated icon font to my local device. Last but not least, I need to copy the generated CSS code to CSS files. This entire process requires a lot of time and effort, but due to time constraints, the process is not compatible in industry.

One developer D2 from Alibaba described how limit the time in their UI development:

Every year, Alibaba has more than 240 events which stores offer special discount, such as Double 11 Global Delight Event, Tmall Thanksgiving Day Event, 1212 Global Discount Event, etc. Due to the high demand for the UI development in the duration of events, we are required to implement UIs in 3 or 4 days.

Developers also considered the trade-off between UI performance and its value. Since the usage of icon font does not provide business value, it is often in a low priority in industry. Even if they knew the benefit of using font, they would not put effort in doing this. For instance, D9 explained:

Although I know the icon font is better compared to icon image, I will not apply this approach in development. I usually have 3 tasks in a week, such as UI implementation, bug testing, algorithm implementation, etc. I agree that icon font can improve UI rendering performance and provide better user experience. But, the overall functionality will still work without icon font. In contrast, without bug testing, the front-end codes may not work, resulting in significantly impact on the company business. And if I do not implement the algorithm, other developers will not be able to apply the API in their development, which will slow down the development speed and delay the product release time.

Another example shows the potential gap between industry and individual is that 50% developers mentioned that they use font to render icon image when developing their personal projects, such as homepage, blog, tutorial, etc. For example, D2 said,

When I developed my first personal website, I discovered Font Awesome [27], a font toolkit to render icons by simply adding class description. Since this is my website, I can design freely according to my preferences. To quickly develop my website, I used the font in Font Awesome to implement all the icon images in my website. However, it is not applicable in industry. In industry, every icon is well designed according to the company culture and design specifications. Therefore, it is not suitable to apply widely used icon font resources from online platforms. In addition, using online icon fonts involves intellectual property (IP) issues which must be avoided in the industry.

> Despite most of the developers know the benefits of using font to render icons, few of them implement font in practice. The icon they used is distinct to the online resources as it comprises company culture and design guidelines. Therefore, rather than directly using the online resources, developers have to spend extra effort in converting icon images to font, which is time-consuming and laborious.

3.2 Research Question 2: do developers write descriptions for icon images?

All of them mentioned that they did write descriptions/comments in their personal projects, such as assignments, homepage, etc. However, half of developers did not write descriptions in practice due to the following practical reasons. First, since the readability of code is not a mandatory requirement, many developers did not write well-formatted descriptions for code. For example, the code in the industry cannot be released as open-source. As D9 said that *"Since our code can not be released to public, I would not spend too much time on writing comments in code because only a few internal developers would collaborate on my tasks."* In addition, since updating iteration in the industry is fast, it is not worth to put too much effort in commenting, especially for icons. For example, D10 said, *In the year of 2019, our company developed over 1 million UIs. Due to the diversity of UIs, few designs are re-implemented and few code are reviewed. Because of the fast updating iteration and low reusing rate, I did not write well-formatted comment, particularly for the images. I was developing a shopping application which images cover more than half of the UIs. To develop the large amount of images quickly, I prefer using <img> tag without any alternative description.*

Second, 80% developers mentioned that writing a well understood description is a challenging task. It requires developers to understand the intention of the icons, while few developers pay attention to the content of the UIs. For example, D7 explained,

I agree that the clear descriptions in the code can keep the code readable and "save lives", while unreasonable descriptions "kill lives". However, it is hard to write a good description. Here is the process of how I write the descriptions: Firstly, I design a comment for every component, image, ..., based on its characteristic. Secondly, I rename and simplify the comments according to practical requirement. Thirdly, I check if the comments match the content of UIs or not. Then I repeat this process until the deadline. And obviously, the process is time-consuming and not applicable in the industry.

Despite the insufficient descriptions in the code may not impede professional developers, it creates a significant cognitive burden for interns and new developers. For example, D3 said,

I am a junior student who came to the company for internship. The first task assigned to me by my leader was to understand the code. However, I found that most of the code is uncommented, which makes it very difficult for me to understand. To understand this part of code, I asked more than 5 developers who participated this project. These uncommented codes negatively influenced my work.

> Developers rarely write descriptions for images, especially for icons, because the loose restriction on code readability makes developers less cared about code descriptions. Most of developers agree with difficulty on designing simple, concise and easy-understood descriptions. The lack of description can adversely affect novice employees and lead to inadequate understanding of the code.

4 APPROACH

Numerous online icon design sharing websites such as Font Awesome [27], Google Material Design Icons [31], provide comprehensive icon library to assist designers and developers in designing and coding. In these online icon websites, each label matches only one icon. While in real case, one label may have several different designs, revealing the limited diversity of these websites. In this work, we select Alibaba Iconfont website [2] as our study subject - not only because it has gained significant popularity among designers' community, but also due to it has became repositories of knowledge with millions of diverse icon designs created by designers.

To collect icon images and associated labels, we built a web crawler based on the Breadth-First search strategy [55] i.e., collecting a queue of URLs from a seed list, and putting the queue as the seed in the second stage while iterating. The crawling process continued from December 12, 2019 to July 1, 2020 with a collection of 100k graphical icon images. We first carried out an empirical study of collaborative icon labelling in Alibaba Iconfont to understand its characteristics for motivating the required tool support (Section 4.1). To solve RQ1, we proposed an automated convertion technique, taking an icon image as the input, and outputting a vector graphics font(Section 4.2). Then, to address RQ2, we got inspired from the findings in our empirical study to develop a deep learning model to automatically assign the classes of icons in order to reduce the effort of manually designing the description of icons (Section 4.3). Additionally, we applied a primary color detection method based on computer vision to keep track of the primary color of the icon image in order to support more detailed description in code (Section 4.4).

4.1 Empirical Study

During the process of open coding the categories of icons semantic, we find that one label can be written in different styles. For example, the label "crop" can be written in not only its standard format, but also its derivations synonyms like "prune", "clip", "crop-tool". Moreover, due to the icon image labelling process in Iconfont is informal and icon designs are contributed by thousands of designers with very diverse technical and linguistic backgrounds the same concept may labeled in many user defined terms such as "crop-fill", "crop-portrait", "icon-crop-solid-24px". The wide presence of forms poses a serious challenge to icon classification task. For example, the icon can be described to the class of "crop" or "clip", which makes sense in both classes.

To address the problem, we adopted association rule mining [1] to discover label correlations from label co-occurrences in icons. We leveraged the visual information from the icon images and textual information from the labels to group a pair-wise correlation of labels. For measuring the visual similarity, we adopted the image similarity score MSE [71] $sim_{vis(x,y)}$ to calculate the likelihood if two icons are the same. For measuring the textual information, we first trained a word embedding [53] model to convert each label into a vector that encodes its semantic. Then we defined a lexical similarity threshold based on the string edit distance [44] $sim_{text(x,y)}$ to check if two labels are similar enough in the form. The labels are grouped as a pair-wise correlation if $sim_{vis(x,y)} \geq 0.9$ or $sim_{text(x,y)} \geq 0.9$. As we wanted to discover the semantics and construct a lexicon of categories, we found frequent pairs of labels. A pair of labels is frequent if the percentage of how many icon images are labelled with this pair of tags compared with all the images is above the minimum support threshold $t_{sup} \geq 0.001$. Given a frequent pair of labels $\{t_1, t_2\}$, association rule mining generated an association rule $t_1 \Rightarrow t_2$ if the confidence of the rule $t_{conf} \geq 0.2$. Given the mined association rules, we constructed an undirected graph $G(V, E)$, where the node set V contains the labels appearing in the association rules, and the edge set E contains undirected edges $< t_1, t_2 >$ (i.e., pair of label associations) if the two labels have the association $t_1 \Rightarrow t_2$ or $t_2 \Rightarrow t_1$. Note that the graph is

Table 1: The 40 icon classes identified through an iterative open coding of 100k icons from the Iconfont [2].

CLASS	ASSOCIATED LABEL	EXAMPLES	NUMBER
add	plus, addition, increase, expand, create		357
calendar	date, event, time, planning		324
camera	photo, take-photo		355
chat	chat-bubble, message, request, comment		372
complete	finish, confirm, tick, check, ok, done		432
computer	laptop, device, computer-response, desktop		521
crop	prune, crop-tool, shear, clipper crop-portrait		436
download	file-download, save, import, cloud		444
edit	editing, handwriting, pencil, pen, edit-fill, modify		546
emoji	amojee, sad, happy, emotion		374
envelope	letter, email, mail, inbox		332
exit	quit, close, switch-off, logout		404
flower	flowers, flower pot, sunflower, valentine-flower		377
gift	present, reward, surprise		340
house	home, rent, house-area, house asset, building, mall		378
left	return, back, prev, backwards		531
like	thumb-up, heart, vote, hand-like, upvote, dislike, favourite		386
location	gps, direction, compass, navigation		543
menu	menu file, card, menufold, menu-line, more, dashboard		351
minus	remove, minus (with circle), minus-sign		556
music	music-note, music-library, musical-instrument		375
news	newspaper, info, announcement		423
package	package-up, package-sent, handpackage, personal package		362
pay	money, wallet, dollar, commerce		364
person	user, avatar, account, customer		562
photo	image, picture, camera		481
play	playicon, broadcast, play voice, play button, play arrow		498
question	ask, faq, information, help, info, support		350
refresh	reload, sync, reset, recreate		321
right	forward, next, go, arrow-forward		412
safe	safe box, safety, safety certificate, lock, secure		476
search	investigate, search-engine, magnifier, find, glass		377
send	send-arrow, paper-plane, message		318
settings	toolbox, gear, preferences, options		317
shopping	cart, shopping-bag, checkout		472
signal	signal-tower, wave, radio, broadcast		548
sound	speaker, sound volume, player		415
star	collection, rate, favourite		424
switch	switch-on/off, switcher, open, close		319
text	word, textbox, font, size		446
visibility	visible, show, hide, visibility-off, in-sight		339
warn	alarm, warning, error, report, alert		369
wifi	wi-fi, wireless, network, signal		429
zoom-in	fullscreen, expand, adjust, magnifier		384

undirected because association rules indicate only the correlations between antecedent and consequent. All threshold values were carefully selected through manually check, considering the balance between the information coverage and overload.

To identify the set of frequently occurred icon label categories, we performed an iterative open coding of most frequent co-occurring labels (or approximately 9.2% of the dataset 542,334 in total) with existing expert lexicon of categories in books and websites such as *Google's Material icon set* [31], *IBM's Design Language of Iconography* [37] and *Design Pattern Gallery* [56]. Two researchers from our team independently coded the categories of these labels, noting any part of the initial vocabulary. Note that both researchers have design experiences in both icon images and UI development.

After the initial coding, the researchers met and discussed the discrepancies and the set of new label categories until consensus was reached. A semantic icon categories can be seen in Table 1. We observed two distinct characteristics in icon images compared to the physical-world objects.

High cross-class similarity: Icon images of different classes often have similar size, shape and visual features. The visual differences to distinguish different classes of icons can be subtle, particularly small widgets are differentiated by small visual cues. For example, the difference between "newspaper" and "file" lies in a text of news at the top/bottom side of "newspaper", while a plus/minus symbol distinguishes "zoom_in"/"zoom_out" from "search". In addition, direction is also an important aspect to distinguish classes. For example, the inclined waves represent "signal" and the upward waves represent "wifi". Existing object classification tasks usually deal with physical objects with distinct features across classes, for example, fishes, flowers, hockey and people in the popular ImageNet dataset [21]. High cross-class similarity affects classification as the class can be not easily distinguished.

Small, transparent and low contrast: To make UI unique and stylish in the screen, icons are usually small and partially transparent, such as the last icon image in the "minus" class shown in Table 1. The transparent icons in the UIs do not cause vision conflict, while they are less visible when separated from the background context. For example, the first icon in the "text" class in Table 1 is an icon with low color contrast and uses transparency and shadow to stress contrast. While the contrast of the object is obvious in the current dataset, especially apparent in the greyscale format such as MNIST dataset [43].

> Existing icon sharing sites contain a wide presence of forms of labeling. Based on different background knowledge, designers use different same-meaning labels to annotate the same icon. Such limitation not only confirms our finding of difficulty of commenting in Section 3.2, but also hinders the potential challenge in classification task. Therefore, a data mining approach capturing visual and textual information is applied to construct a lexicon in icons. By observing the lexicon, we find that two distinct characteristics of icon different from the existing physical object orientated dataset.

4.2 Font Convertion from Icon Image

Unlike converting font to image, transcribing image to font, which is also known as image tracing problem, is a difficult task. In this work, we adopted the state-of-the-art Potrace [64] in Figure 1A. We first applied a pre-processing method for converting color to binary (i.e., black and white) image by setting a threshold to control the bit of each pixel after calculating the average value in three channels $(R+G+B)/3$. We regarded the pixel is of white if the average value is larger than 128, while black if the value is equal to or smaller than 128. Then, we detected the edge in the black-white image. An edge is defined to be a border between a white pixel and a black pixel, which indicate which pixels from the original image constitute the borders of region. Note that the edge is assigned a direction so that when moving from the first endpoint to the second, the black pixel is on the left (as shown in Figure 1A edge detection). This process was repeated until we reached the starting point, at which point we have found a closed path which encloses a black region. Once the border was found, we approximated/optimized the border with a polygon to figure out which border pixels is possible to connect with a straight line such that the line passes through all the border pixels between its endpoints. To detect the optimal polygon, we computed a penalty value to measure the average distance from the edge to the pixels it approximates. The polygon with the smallest penalty (equivalent to the polygon with the fewest pixels) is the optimal one. Finally, we used a cubic curve defined by four control points (also known as Bezier curve [63]) to smooth the corners. The first and fourth control points (i.e., midpoints of the edges of the polygon) give the locations of the two endpoints of the curve, while the second and third (i.e., chosen on the polygon edges through the endpoints) indicate the direction and magnitude of the derivative of the curve at each endpoint.

4.3 Prediction for Icon Image

Traditional Convolutional Nerual Network (CNN) [42, 43] has shown great potential as a solution for difficult vision problems. MobileNetV2 [62] distills the best practices in convolutional network design into a simple architecture that can serve as competitive performance but keep low parameters and mathematical operations to reduce computational power. The architecture of the network is shown in Figure 1B.

Instead of using regular convolutional layers widely used in traditional CNN architectures to capture essential information from images but are expensive to compute, MobileNetV2 adopted a more advanced one, depthwise separable convolutions. Depthwise separable convolution combined a $3 * 3$ convolution layer and two $1 * 1$ convolution layers. The $1 * 1$ convolution layer (also named as pointwise convolution layer) was used to combine the filter values into new features, while the $3 * 3$ convolution (also called as depthwise convolution layer) was used to filter the input feature map. Inspired from the dimension augmentation in the work of [46], MobileNetV2 used a $1 * 1$ pointwise convolution layer to expand the number of channels in the input feature map. Then it used a $3 * 3$ depthwise convolution layer to filter the input feature map and a $1 * 1$ convolution layer to reduce the number of channels of feature map. The network borrowed the idea of residual connection in ResNet [33] to help with the flow of gradients. In addition, batch normalization and activation layer were added between each depthwise convolution layer and pointwise convolution layer to make the network more stable during training. For detailed implementation, we adopted the stride of 2 in the depthwise convolution layer to downsample the feature map. For the first two activation layers, the network used ReLU6 defined as $y = min(max(0, x), 6)$ because of its robustness in low-precision computation [35], and a linear transformation (also known as Linear Bottleneck Layer) was applied to the last activation layer to prevent ReLU from destroying features.

4.4 Color Detection of Icon Image

Since the convertion between icon image and font sacrifices the color identity, we added an attribute to keep track of the primary

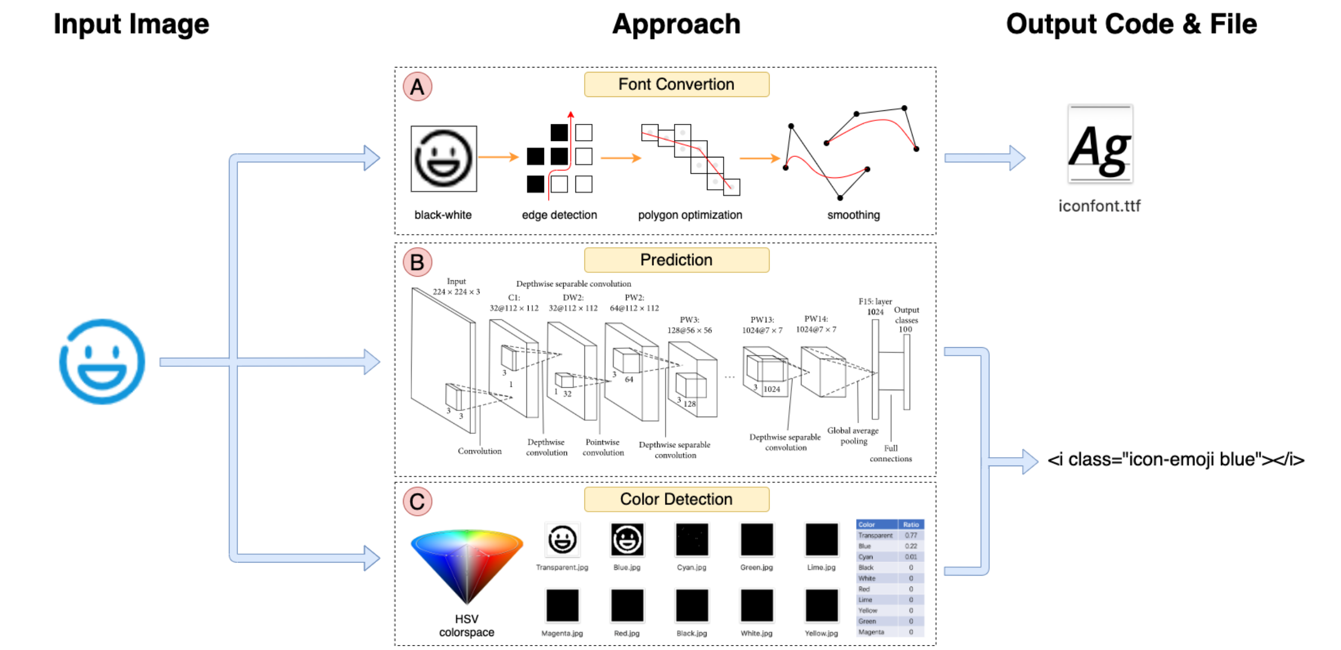


Figure 1: The approach of our tool, *Auto-Icon*, involving font convertion, prediction and color detection.

color of the icon image. To that end, we adopted HSV colorspace for color detection. We first removed the fourth alpha channel as transparent and made a conversion from RGB color to HSV colorspace. Each RGB color has a range of HSV vale. The lower range is the minimum shade of the color that will be detected, and the upper range is the maximum shade. For example, blue is in the range of $\langle 100, 43, 46\rangle - \langle 124, 255, 255\rangle$. Then, we created a mask for each color (black, blue, cyan, green, lime, megenta, red, white) as shown in Fig. 1C. The mask is the areas that HSV value on pixels match the color between the lower range and upper range. Finally, we calculated the area of the mask in each color and the corresponding image occupancy ratio. The color with the maximum ratio was identified as the primary color of the icon image (the blue in the example in Fig. 1C).

5 EXPERIMENTS

In this section, we first set up an experiment to analyze the performance of our model in Section 4.3. Then we conduct a pilot user study to evaluate the usefulness of our tool. Furthermore, we demonstrate its usefulness on a large-scale industrial benchmark. The goal of our experiments is to answer the following research questions, in terms of accuracy, efficiency and applicability. *RQ 1: how accurate is our model in predicting labels for icon images? RQ 2: how much do our tool increase the efficiency of UI development? RQ 3: what are the developers' opinions on the usability of our tool?*

5.1 Icon Prediction

5.1.1 Dataset: We leveraged the categorization during the creation of the semantic vocabulary (in Table 1), and corresponding icon images and attached labels as the training data. The foundation of the deep learning model is the big data, so we only selected categories with frequency larger than 300 for training the model. Therefore, there are 100 categories left with the number of icon images ranging from 311 to 589. Given all the data assigned to each label, we randomly split these 41k icon images into train/validation/test dataset with a ratio of 8:1:1 (33K:4K:4k).

5.1.2 Baselines: We set up several basic machine-learning baselines including the feature extraction (e.g., color histogram [69], scale-invariant feature transform [50]) with machine-learning classifiers (e.g., decision tree [59], SVM [19]). Apart from these conventional machine learning based baselines, we also set up several derivations of state-of-the-art deep learning models as baselines to test the importance of different inputs of our approach including backbones (ResNet [33], VGG [65], MobileNet [62]), different input channels (RGB, RGBA). The training and testing configurations for these baselines were the same.

5.1.3 Results: As we trained a classifier to predict label for icon image, we adopted the accuracy as the evaluation metric for the model, illustrated in Table 2. The traditional machine learning method based on the human-crafted features can only achieve about 0.6 average accuracy. Deep learning models perform much better than the best old fashioned methods, i.e., with the 0.3033, 0.29712, 0.2958 increase for ResNet-50, VGG-16, and MobileNetV2 respectively. Although ResNet model performs the best in icon image classification task, it requires relatively long time for prediction (26.535ms per icon) which strongly violates the performance of UI rendering (16ms). In contrast, our model MobileNet is nearly as accurate as ResNet with a performance lag of 0.67%, while being 34.1% faster. And also, we find that the increase of a fourth alpha channel (RGBA)

Method	Histo +SVM	Histo +DT	SIFT +SVM	SIFT +DT	ResNet-50	VGG-16	MobileNetV2 (RGBA)	MobileNetV2 (RGB)
Accuracy	0.5657	0.3267	0.5806	0.4686	**0.8839**	0.8764	0.8348	0.8772
Time (ms)	0.103	0.152	1.702	1.941	26.535	27.282	17.567	**17.485**

Table 2: Label classification accuracy and time estimation in different methods.

decreases the accuracy from 0.8772 to 0.8348, due to two main reasons. First, the result shows that the model with RGB input has a loss value of 0.7844 at epoch 200, which is better than the model with RGBA (0.9231). This is because the supplemented channel greatly increases the parameters of the model, which leads to a decline in the ability of gradient training at the same epochs. Second, based on the principle of optics, the fourth alpha channel does not reflect the morphological characteristics of the image. It is used to reduce information of other three channels by adjusting their color/degree, causing less information captured through training process.

5.2 User Study

5.2.1 Procedures for User Study. 10 developers, all proficient in UI development and have at least 1-year experience, were recruited for this study. We randomly selected five icon images from the real-life UI designs and asked each participant to develop them. To guarantee the participants can objectively develop the icons, we asked whether they have prior knowledge on the icon images (such as development experience, design experience, etc.). The time of the development were recorded. To be fair, participants did not know we were recording the time as the time pressure may affect their development (in quality, speed, etc.) [5, 52]. We set the manual development as the control group. Then, we also asked them to develop five other icon images with the help of our tool which not only automatically convert the image to icon font, but also provide the description (predicted label and color). We called this the experimental group. The detailed developments of two groups for icon images are shown in Table 3.

We then recruited another 10 developers, and each of them was assigned the developments from two control groups and one experimental group. Note that they did not know which one if from the experimental or control group, and for each icon image, we randomly shuffled the order of candidates to avoid potential bias. Given each development, they individually marked it as readable or not in five-point likert scale (1:not readable at all and 5:strongly readable). To evaluate the performance of usability, we also asked participants to rate how likely they would like to use the development in practice (Acceptable). The measurement is also in five-point scale.

5.2.2 Results: Box plot in Figure 2 shows that the time spent on the development of icon images in the experimental group is much shorter than that in the control group (with an average of 6.05s versus 17.39s, saving 65.2% of time). That is the biggest strength of our tool i.e., developers can quickly develop an icon image by providing descriptions and a font pattern. On average, the overall readability ratings for the experiment group is 4.16, which are significantly higher (48.5%) than the control group (2.8) in Figure 2. Most developers admit that our tool can provide more acceptable results for them. In other words, 94% (4.7/5.0) of developers hope to develop the icon images with the help of our tool in their real development environment compared to 3.96 in the control group. To understand the significance of the differences, we carry out the Mann-Whitney U test [24] (specifically designed for small samples) on the readability and acceptability ratings between the experiment and the control group respectively. The test results suggest that our tool does significantly outperform the baseline in term of these metrics with $p < 0.01$ or $p < 0.05$.

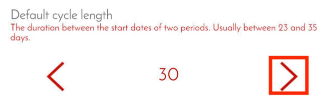
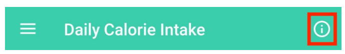
For some icons, the developer gives very low acceptability score to the labels. According to our observation and further survey, we summarise two reasons accounting for those bad cases. (1) Albeit the good performance of our model, we still make wrong predictions for serendipitous icons. Based on the context of icon image, the same icon can have different meaning. For example, the icon image in Figure 3 represents the meaning of "information" in the common case, but consider the text on the right, the meaning of the icon image should be "glossary"/"dictionary". (2) Developers admit the usefulness of converting images to font for providing faster rendering speed. However, they also point out the limitation of replacing image with font. Font is not fully compatible in all browsers and devices. One developer mentioned that they need to make sure that the development works on old devices, in which they usually need to give up latest efficient methods, such as iconfont.

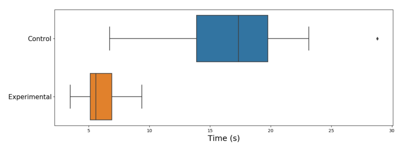
5.3 Industrial Usage

We cooperate our tool with the *Imgcook* platform[3] developed by Alibaba, an intelligent tool to automatically generate front-end codes from UI design files. *Imgcook* has attracted a lot of attention in the community which has a large user base (15k) and generates over 40k UIs. *Auto-Icon* is integrated with the internal automated code generation process and is triggered whenever the design files contain an icon.

In order to evaluate the usability of our tool, we set up a code review metric for measuring the code modification for icon images. Note that the code modification contains multiple contents, such as text, button, etc, we only measure the modification if the object is icon to reduce the potential bias. We adopt a case-insensitive BLEU (BiLingual Evaluation Understudy) [58] as the metric to evaluate the preservation of code. BLEU is an automatic evaluation metric widely used in code difference studies. It calculates the similarity of machine-generated code and human-modified reference code (i.e., ground truth) as $BLEU = BP * exp(\sum_{N}^{n=1} w_n log p_n)$ where p_n denotes the precision, i.e., the ratio of length n token sequences generated by our method are also present in the ground-truth; w_n denotes the weight of different length of n-gram summing to one;

Table 3: Examples of development for icon images in Experimental and Control groups.

	**E**	<i class="icon-left red"></i>
	C1	
	C2	
	C3	
	**E**	<i class="icon-information white"></i>
	C1	<svg class="icon" aria-hidden="true"> <use xlink:url="icon-information"></use> </svg>
	C2	
	C3	<SvgIcon name="white">



Measures	Control	Experimental
Time (s)	17.39	6.05*
Readable	2.8	4.16*
Acceptable	3.96	4.7**

Figure 2: The comparison of Experimental and Control groups. ***denotes** $p < 0.01$, ****denotes** $p < 0.05$.

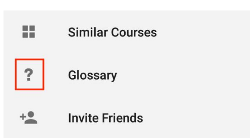


Figure 3: The icon description varies by context.

BP is 1 if c > r, otherwise $e^{(1-r/c)}$ where c is the length of machine-generated sequences and r is the length of ground-truth. A high BLEU score indicates less modification in the code review.

We run the experiment in *Imgcook* with 6,844 icons in 2,031 UIs from August 20, 2020 to September 20, 2020. Among all the testing UI developments, the generated code for icon image reaches 84% BLEU score, which means that most of the code is used directly without any modification. It demonstrates the high usability of *Auto-Icon* in practice. Based on the inspection results, we categorize the modification into four categories. Two reasons are discussed in the Section 5.2.2, in terms of wrong prediction and compatibility concern. There are another two modifications mainly due to industrial practice. First, in order to maintain the consistency of company's coding experience, some developers modify to a prescribed naming/rendering method, for example, packing the icon of "icon-camera" to a <Icon-Camera> tag. Second, UI dynamically changes in practice. Once an element in the UI is changed, the attribute of icon may change, such as color and font size.

Overall, our method achieves 87.7% accuracy in the label prediction for icon images (RQ1). In the survey of 10 developers, we improve the efficiency of developing time and code readability by 65.2% and 48.5%, respectively (RQ2). The majority (4.7/5.0) of the interviewed developers acknowledges the usability of the generated code for icon image by our method, and it is further confirmed in the practice of *Imgcook* with 84% BLEU score (RQ3).

6 DISCUSSION

On developers: Developing icon images in the UIs is a challenging and time-consuming task, even for professional developers. On the one hand, UI developers must enhance performance. Poor development has an adverse effect on the performance of the site. The performance issues comprise a multitude of factors like rendering speed, reusability & flow of the code, etc. On the other hand, UI developers must write a clean, high quality code which can be easily understood and maintained. Inspired by the high performance of font rendering, our work designs an automated method to convert icon image to icon font using computer vision techniques to trace the edge of icon and using graphic algorithm to optimize the edge. In addition, compared with the missing descriptions in the development or brainstorming suitable names which is limited to several developers in the physical world, our deep learning and computer vision techniques based method can quickly identify the label and the color of icon image. Our method once made accessible to developers, can very well help developers achieve efficient icon image coding.

On the generalization of our method: We report and analyze the performance of our CNN-based model for predicting icon image labels in Section 5.1.3. One limitation with our model is that we currently only test our model on 100 labels with enough corresponding icon images. With the cooperation with *Imgcook* platform, the icons in the UI images are a gold resource. First, the icon images are relatively unique, otherwise, developers can reuse the online resources directly. These unique icon images can significantly increase the amount of data, consequently improving the accuracy of our model. Second, developers may modify the description to a serendipitous label which can augment the labels and generalize a broader range of icon descriptions. Due to the time limit, we only collect a small amount of icon images from *Imgcook*. However, we have seen some interesting icon images that do not exists on online sharing platforms and they may improve the generalization of our method.

Area of improvements: Currently, we only predict the label based on icon itself. As discussed in Section 5.2.2, the meaning of icon image varies in different context. To address this problem, we can consider the entire UI, capturing all the related information to make the final prediction. Developers praise the idea of adding descriptions to the code which is a tedious task for them. They wonder whether our model can extend to other elements. An developer hope us to support description for buttons as he finds many buttons do not have descriptive texts to explain its intention, resulting in a bad user experience. We believe our model could help developers in this case as it will not be difficult to extend to other elements once we obtain enough data for the training. Moreover, developers envision the high potential in being able to add icon size descriptions as one of the biggest strength of icon font is lossless scalability. To that end, we can measure the height of the icon and map it to the corresponding font size.

7 CONCLUSION

In this paper, we present a deep-learning and computer vision approach that can provide developers with intelligent support to reduce the development time of icon design in the UI. The core techniques are three-fold. First, we develop an automated image convertion method to turn an icon image into a font in which improving UI rendering speed. Second, to assist developers with better code accessibility, we adopt a deep learning model to automatically predict the descriptive label that convey the semantics of the icon image. Third, according the colorspace of the image, we detect the primary color of the icon to provide developers more knowledge on the image. Our method is incorporated into existing automated code generation platform to extend them beyond effective and descriptive coding.

ACKNOWLEDGMENTS

We appreciate Yanfang Chang and Jiawen Huang for designing and conducting a survey experiment, and all the participants from Imgcook team in Alibaba Group for taking surveys.

REFERENCES

[1] Rakesh Agrawal, Ramakrishnan Srikant, et al. 1994. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, Vol. 1215. 487–499.
[2] Alibaba. 2020. Iconfont+. https://www.iconfont.cn/. Accessed: 2020-09-28.
[3] Alibaba. 2020. Imgcook. https://imgcook.taobao.org/. Accessed: 2020-09-29.
[4] Android. 2019. Systrace. https://developer.android.com/studio/profile/systrace/navigate-report. Accessed: 2020-09-01.
[5] Robert D Austin. 2001. The effects of time pressure on quality in software development: An agency model. *Information systems research* 12, 2 (2001), 195–207.
[6] Abhijeet Banerjee, Hai-Feng Guo, and Abhik Roychoudhury. 2016. Debugging energy-efficiency related field failures in mobile apps. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*. 127–138.
[7] Abhijeet Banerjee and Abhik Roychoudhury. 2016. Automated re-factoring of android apps to enhance energy-efficiency. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*. 139–150.
[8] Michel Beaudouin-Lafon. 2004. Designing interaction, not interfaces. In *Proceedings of the working conference on Advanced visual interfaces*. 15–22.
[9] Alison Black. 2017. Icons as carriers of information. *Information design: Research and practice* (2017), 315–329.
[10] Margaret Butler. 2010. Android: Changing the mobile landscape. *IEEE pervasive Computing* 10, 1 (2010), 4–7.
[11] Antonin Carette, Mehdi Adel Ait Younes, Geoffrey Hecht, Naouel Moha, and Romain Rouvoy. 2017. Investigating the energy impact of android smells. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 115–126.
[12] Chunyang Chen, Sidong Feng, Zhengyang Liu, Zhenchang Xing, and Shengdong Zhao. 2020. From Lost to Found: Discover Missing UI Design Semantics through Recovering Missing Tags. *arXiv preprint arXiv:2008.06895* (2020).
[13] Chunyang Chen, Sidong Feng, Zhenchang Xing, Linda Liu, Shengdong Zhao, and Jinshui Wang. 2019. Gallery DC: Design Search and Knowledge Discovery through Auto-created GUI Component Gallery. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–22.
[14] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhut, Guoqiang Li, and Jinshui Wang. 2020. Unblind your apps: Predicting natural-language labels for mobile GUI components by deep learning. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 322–334.
[15] Jieshan Chen, Mulong Xie, Zhenchang Xing, Chunyang Chen, Xiwei Xu, and Liming Zhu. 2020. Object Detection for Graphical User Interface: Old Fashioned or Deep Learning or a Combination? *arXiv preprint arXiv:2008.05132* (2020).
[16] Sen Chen, Lingling Fan, Chunyang Chen, Ting Su, Wenhe Li, Yang Liu, and Lihua Xu. 2019. Storydroid: Automated generation of storyboard for Android apps. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 596–607.
[17] Sen Chen, Lingling Fan, Chunyang Chen, Minhui Xue, Yang Liu, and Lihua Xu. 2019. GUI-Squatting Attack: Automated Generation of Android Phishing Apps. *IEEE Transactions on Dependable and Secure Computing* (2019).
[18] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li. 2018. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *computers & security* 73 (2018), 326–344.
[19] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.
[20] Tobias Dehling, Fangjian Gao, Stephan Schneider, and Ali Sunyaev. 2015. Exploring the far side of mobile health: information security and privacy of mobile health apps on iOS and Android. *JMIR mHealth and uHealth* 3, 1 (2015), e8.
[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.
[22] Android Developers. 2020. Inspect GPU rendering speed and overdraw. https://developer.android.com/topic/performance/rendering/inspect-gpu-rendering. Accessed: 2020-09-28.
[23] Android Developers. 2020. Test UI performance. https://developer.android.com/training/testing/performance. Accessed: 2020-09-28.
[24] Michael P Fay and Michael A Proschan. 2010. Wilcoxon-Mann-Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics surveys* 4 (2010), 1.
[25] Ruitao Feng, Sen Chen, Xiaofei Xie, Lei Ma, Guozhu Meng, Yang Liu, and Shang-Wei Lin. 2019. Mobidroid: A performance-sensitive malware detection system on mobile platform. In *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 61–70.
[26] Fontello. 2020. icon fonts generator. http://fontello.com/. Accessed: 2020-09-07.
[27] Inc. Fonticons. 2020. Font Awesome. https://fontawesome.com/. Accessed: 2020-09-28.
[28] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. 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.
[29] Yi Gao, Yang Luo, Daqing Chen, Haocheng Huang, Wei Dong, Mingyuan Xia, Xue Liu, and Jiajun Bu. 2017. Every pixel counts: Fine-grained UI rendering analysis for mobile applications. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.
[30] María Gómez, Romain Rouvoy, Bram Adams, and Lionel Seinturier. 2016. Mining test repositories for automatic detection of UI performance regressions in Android apps. In *Proceedings of the 13th International Conference on Mining Software Repositories*. 13–24.
[31] Google. 2020. Material Icons. https://material.io/resources/icons/. Accessed: 2020-09-09.

[32] Chris Harrison, Gary Hsieh, Karl DD Willis, Jodi Forlizzi, and Scott E Hudson. 2011. Kineticons: using iconographic motion in graphical user interface design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* 1999–2008.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 770–778.

[34] Geoffrey Hecht, Omar Benomar, Romain Rouvoy, Naouel Moha, and Laurence Duchien. 2015. Tracking the software quality of android applications along their evolution (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 236–247.

[35] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[36] Gang Huang, Mengwei Xu, Felix Xiaozhu Lin, Yunxin Liu, Yun Ma, Saumay Pushp, and Xuanzhe Liu. 2017. Shuffledog: Characterizing and adapting user-perceived latency of android apps. *IEEE Transactions on Mobile Computing* 16, 10 (2017), 2913–2926.

[37] IBM. 2020. Design Language. https://www.ibm.com/design/language/iconography/ui-icons/library. Accessed: 2020-09-09.

[38] IcoMoon. 2020. Icon Font & SVG Icon Sets. https://icomoon.io/. Accessed: 2020-09-07.

[39] Muhammad Nazrul Islam. 2015. Exploring the intuitiveness of iconic, textual and icon with texts signs for designing user-intuitive web interfaces. In *2015 18th International Conference on Computer and Information Technology (ICCIT).* IEEE, 450–455.

[40] Bridgett A King and Norman E Youngblood. 2016. E-government in Alabama: An analysis of county voting and election website content, usability, accessibility, and mobile readiness. *Government Information Quarterly* 33, 4 (2016), 715–726.

[41] Charalambos Koutsourelakis and Konstantinos Chorianopoulos. 2010. Icons in mobile phones: Comprehensibility differences between older and younger users. *Information Design Journal* 18, 1 (2010), 22–35.

[42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems.* 1097–1105.

[43] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[44] Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, Vol. 10. 707–710.

[45] Wenjie Li, Yanyan Jiang, Chang Xu, Yepang Liu, Xiaoxing Ma, and Jian Lü. 2019. Characterizing and detecting inefficient image displaying issues in Android apps. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER).* IEEE, 355–365.

[46] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network in network. *arXiv preprint arXiv:1312.4400* (2013).

[47] Mario Linares-Vasquez, Christopher Vendome, Qi Luo, and Denys Poshyvanyk. 2015. How developers detect and fix performance bottlenecks in android apps. In *2015 IEEE international conference on software maintenance and evolution (ICSME).* IEEE, 352–361.

[48] Yepang Liu, Chang Xu, and Shing-Chi Cheung. 2014. Characterizing and detecting performance bugs for smartphone applications. In *Proceedings of the 36th international conference on software engineering.* 1013–1024.

[49] Yepang Liu, Chang Xu, and Shing-Chi Cheung. 2015. Diagnosing energy efficiency and performance for mobile internetware applications. *IEEE Software* 32, 1 (2015), 67–75.

[50] David G Lowe et al. 1999. Object recognition from local scale-invariant features.. In *iccv*, Vol. 99. 1150–1157.

[51] Muhammad Noman Malik, Huma Hayat Khan, and Fazli Subhan. 2017. Sustainable Design of Mobile Icons: Investigating Effect on Mentally Retarded User's. *Journal of Medical Imaging and Health Informatics* 7, 6 (2017), 1419–1428.

[52] Mika V Mäntylä, Kai Petersen, Timo OA Lehtinen, and Casper Lassenius. 2014. Time pressure: a controlled experiment of test case development and requirements review. In *Proceedings of the 36th International Conference on Software Engineering.* 83–94.

[53] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems.* 3111–3119.

[54] Higinio Mora, Virgilio Gilart-Iglesias, Raquel Pérez-del Hoyo, and María Dolores Andújar-Montoya. 2017. A comprehensive system for monitoring urban accessibility in smart cities. *Sensors* 17, 8 (2017), 1834.

[55] Marc Najork and Janet L Wiener. 2001. Breadth-first crawling yields high-quality pages. In *Proceedings of the 10th international conference on World Wide Web.* 114–118.

[56] Theresa Neil. 2014. *Mobile design pattern gallery: UI patterns for smartphone apps.* " O'Reilly Media, Inc.".

[57] Javad Nejati and Aruna Balasubramanian. 2016. An in-depth study of mobile browser performance. In *Proceedings of the 25th International Conference on World Wide Web.* 1305–1315.

[58] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics.* 311–318.

[59] J Ross Quinlan. 1983. Learning efficient classification procedures and their application to chess end games. In *Machine learning.* Springer, 463–482.

[60] Sanae Rosen, Bo Han, Shuai Hao, Z Morley Mao, and Feng Qian. 2017. Push or request: An investigation of HTTP/2 server push for improving mobile performance. In *Proceedings of the 26th International Conference on World Wide Web.* 459–468.

[61] Anne Spencer Ross, Xiaoyi Zhang, James Fogarty, and Jacob O Wobbrock. 2018. Examining image-based button labeling for accessibility in Android apps through large-scale analysis. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility.* 119–130.

[62] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 4510–4520.

[63] Thomas W Sederberg and Rida T Farouki. 1992. Approximation by interval Bézier curves. *IEEE Computer Graphics and Applications* 5 (1992), 87–88.

[64] Peter Selinger. 2003. Potrace: a polygon-based tracing algorithm. *Potrace (online), http://potrace. sourceforge. net/potrace. pdf (2009-07-01)* (2003).

[65] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[66] Shamik Sural, Gang Qian, and Sakti Pramanik. 2002. Segmentation and histogram generation using the HSV color space for image retrieval. In *Proceedings. International Conference on Image Processing*, Vol. 2. IEEE, II–II.

[67] Fahui Wang. 2012. Measurement, optimization, and impact of health care accessibility: a methodological review. *Annals of the Association of American Geographers* 102, 5 (2012), 1104–1112.

[68] Xu Wang, Chunyang Chen, and Zhenchang Xing. 2019. Domain-specific machine translation with recurrent neural network for software localization. *Empirical Software Engineering* 24, 6 (2019), 3514–3545.

[69] Xiang-Yang Wang, Jun-Feng Wu, and Hong-Ying Yang. 2010. Robust image retrieval based on color histogram of local feature regions. *Multimedia Tools and Applications* 49, 2 (2010), 323–345.

[70] Yan Wang and Atanas Rountev. 2016. Profiling the responsiveness of Android applications via automated resource amplification. In *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft).* IEEE, 48–58.

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

[72] Tianyong Wu, Jierui Liu, Zhenbo Xu, Chaorong Guo, Yanli Zhang, Jun Yan, and Jian Zhang. 2016. Light-weight, inter-procedural and callback-aware resource leak detection for android apps. *IEEE Transactions on Software Engineering* 42, 11 (2016), 1054–1076.

[73] Xusheng Xiao, Xiaoyin Wang, Zhihao Cao, Hanlin Wang, and Peng Gao. 2019. Iconintent: automatic identification of sensitive ui widgets based on icon classification for android apps. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE).* IEEE, 257–268.

[74] Mulong Xie, Sidong Feng, Zhenchang Xing, Jieshan Chen, and Chunyang Chen. 2020. UIED: a hybrid tool for GUI element detection. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 1655–1659.

[75] Shunguo Yan and PG Ramachandran. 2019. The current status of accessibility in mobile apps. *ACM Transactions on Accessible Computing (TACCESS)* 12, 1 (2019), 1–31.

[76] Zhenlong Yuan, Yongqiang Lu, and Yibo Xue. 2016. Droiddetector: android malware characterization and detection using deep learning. *Tsinghua Science and Technology* 21, 1 (2016), 114–123.

[77] Xiaoyi Zhang, Anne Spencer Ross, and James Fogarty. 2018. Robust Annotation of Mobile Application Interfaces in Methods for Accessibility Repair and Enhancement. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology.* 609–621.

[78] Dehai Zhao, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020. Seenomaly: Vision-Based Linting of GUI Animation Effects Against Design-Don't Guidelines. In *42nd International Conference on Software Engineering (ICSE'20). ACM, New York, NY.*

[79] Hui Zhao, Min Chen, Meikang Qiu, Keke Gai, and Meiqin Liu. 2016. A novel pre-cache schema for high performance Android system. *Future Generation Computer Systems* 56 (2016), 766–772.