TapShoe: Modeling Mobile Interface Tappability Using Crowdsourcing and Deep Learning

Anonymous
Anonymous
City, State
E-Mail

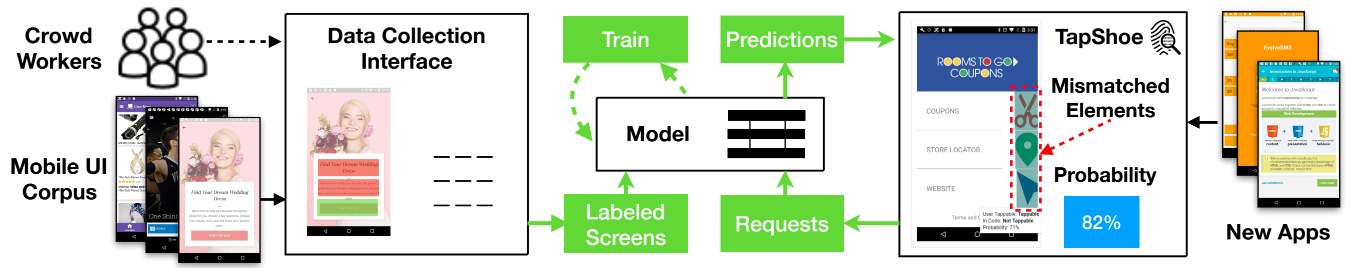


Figure 1: TapShoe learns a deep model from large-scale dataset of mobile tappability collected via crowdsourcing, and predicts tappability of new interfaces for designers and developers to uncover potential usability issues.

ABSTRACT

Tapping is an immensely important gesture in mobile touchscreen interfaces, yet people still frequently are required to learn which elements are tappable through trial and error. Predicting human behavior for this everyday gesture can help mobile app designers understand an important aspect of the usability of their apps without having to run a user study. We present TapShoe, an approach for modeling tappability of mobile interfaces at scale. We conducted large-scale data collection of interface tappability over a rich set of mobile apps using crowdsourcing and computationally investigated a variety of signifiers that people use to distinguish tappable versus not-tappable elements. Based on the dataset, we developed and trained a deep neural network that predicts how likely a user will perceive an interface element as tappable versus not tappable. Using the trained model, we developed an interface that automatically diagnoses mismatches between the tappability of each element as perceived by a human user predicted by our model, and the actual tappable state of the element intended by the developer or designer specified in the interface code or description. TapShoe achieved reasonable accuracy: mean precision 88.38% and recall 89.08% on identifying tappable versus not-tappable elements as perceived by humans. TapShoe is well received by designers via an informal evaluation with 7 professional interaction designers.

CCS CONCEPTS

• **Human-centered computing**;

KEYWORDS

Mobile interfaces; tappability, crowdsourcing, deep learning;

1 INTRODUCTION

Tapping is arguably the most important gesture on mobile interfaces. Yet, it is still difficult for people to distinguish between tappable and not-tappable elements in a mobile interface. This would be less of an issue on traditional desktop GUIs where the style of clickable elements (e.g., buttons) are often conventionally defined. However, with the diverse styles of mobile touchscreen interfaces, tappability has become a crucial usability issue. Poor tappability can lead to a lack of discoverability [25] and false affordances [11] that can lead to user frustration, uncertainty, and errors [2, 3].

Signifiers can [25] indicate to a user how to interact with an interface element. Designers use visual properties of an interface element such as color, depth, and shadows to signify its "clickability" [2] or "tappability" in mobile interfaces. Perhaps the most ubiquitous signifiers in today's interfaces are the blue color and underline of a link, and the design of a button which both strongly signify to the user that they should be clicked. These common interface elements have been learned over time and are well understood as clickable [24]. However, modern platforms for mobile apps frequently introduce new design patterns and interface elements, and designing these to include appropriate signifiers for tappability is challenging. Additionally, mobile interfaces cannot take advantage of many useful clickability cues available in web and desktop interfaces, such as hover states. With the flat design trend, traditional signifiers have been altered, which potentially causes uncertainty and mistakes [3]. More data may be needed to confirm these results, however, we argue that we need more data and automated methods to fully understand the users' perceptions of tappability as design trends evolve over time.

One method that interface designers currently use to understand signifiers in their interfaces is conducting a tappability study or a visual affordance test [4]. However, it is time consuming to conduct such studies. In addition, the findings from these studies are often limited to a specific app or interface design. We aim to understand signifiers at a large scale across a diverse array of interface designs and to diagnose tappability problems in new apps automatically without conducting user studies.

In this work, we present TapShoe, an approach for modeling interface tappability at scale. In addition to acquiring a deeper understanding about tappability, we develop tools that can automatically identify tappability issues in a mobile app interface (see Figure 1). To this end, we trained a deep learning model based on a large dataset of labeled tappability of mobile interfaces that we collected via crowdsourcing. The dataset includes more than 20,000 examples from more than 3,000 mobile screens. The trained model is then served via a web interface that diagnoses mismatches between the tappability of a mobile interface as perceived by the user and its actual tappable state as specified in the interface code or description by the designer or developer. TapShoe achieved reasonable accuracy with mean precision 88.38% and recall 89.08% on identifying tappable verus not-tappable elements. We evaluated TapeShoe with 7 professional designers via an informal interview who were all positive about the use of TapShoe in realistic design situations. The contributions of this paper include the following.

(1) An approach for understanding interface tappability at scale using crowdsourcing and computational signifier analysis, and a set of new findings about mobile tappability;
(2) A deep neural network model that learns human perceived tappability of interface elements from a range of interface features, including the spatial, semantic and visual aspects of an interface element, and an in-depth analysis about the model behavior;
(3) An interactive system that uses the model to examine a mobile interface by automatically scanning the tappability of each element on the interface, and identifies mismatches with their intended tappable behavior.

2 RELATED WORK

The concepts of signifiers and affordances are integral to our work as our aim was to capture them in a structured way to construct a predictive model and to understand how they are being used in a large set of real mobile interfaces. Affordances were originally described by [12] as the actionable properties between the world and actor (i.e., person). Don Norman [23, 24] popularized the idea of affordances of everyday objects, such as a door, and later introduced the concept of a "signifier" as it relates to user interfaces [24]. Designers can manipulate signifiers to signify to the user an interface element's perceived affordances. Gaver [11] described how signifiers are used for a button and described how graphical techniques can be used to aid this perception such as shadows and rounded corners. These early works form the core of our current understanding of what makes a person understand what is interactive. By collecting a large scale dataset of tappability labels, we hope to aid our understanding of which signifiers are having an impact at scale.

Since those early works, there have been a few studies about the factors influencing clickability in web interfaces [2, 3]. Usability testing methods have also adopted the idea of visual affordance testing [4] to diagnose clickability issues. However, these studies have been conducted at a small scale and are typically limited to the single app being tested. We are not aware of any large-scale data collection and analysis across app interfaces to enable diagnosis of tappability issues, nor any machine learning approaches that learn from this data to automatically predict the elements that users will perceive as tappable or not tappable.

To identify tappability issues automatically, we needed to collect data on a large scale to allow us to use a machine learning approach for this problem. Recently, data-driven approaches have been adopted to identify usability issues [8], and collect mobile app design data at scale [7, 9]. Perhaps most closely related to our work is Zipt [8] which leverages existing app implementations to enable comparative user performance testing at scale. Zipt uses crowd workers to construct user flow visualizations through apps that can help designers visualize the paths users will take through their app for specific tasks. However, with this approach, designers must still manually diagnose the usability issues by examining the visualizations. Our goal is to be able to identify common usability issues automatically. In this paper, we focus on one of the most important usability issues on mobile interfaces—identifying cases where false affordances or missing signifiers will cause a user to misidentify a tappable or not-tappable interface element.

Like Zipt [8], our work relies on crowdsourcing to collect user data to aid the diagnosis of usability issues. We used Amazon's Mechanical Turk which has previously provided a platform for conducting large-scale usability [22] and human subjects experiments [15, 16, 28], and in gathering data about the visual design of user interfaces [13, 20, 31]. However, our work goes beyond data collection and analysis by developing machine learning models to automate the task of tappability examination.

Deep learning [18] is an effective approach for learning from a large-scale dataset. In our work, we trained a deep feedforward network, which uses convolutional layers for image processing, to automatically predict the tappability of elements as perceived by human users. Recent work has adopted deep learning approaches to automatically analyze and predict human performance on mobile apps for tasks such as grid [27] and menu selection [19]. Deep learning models have also been built to identify salient elements in graphic designs and interfaces [6]. However, no work has yet applied these models to predicting the tappability of interface elements. Deep learning allowed us to leverage a rich set of features involving the semantic, spatial and visual properties of an element without extensive feature engineering.

3 UNDERSTANDING TAPPABILITY AT SCALE

A common type of usability testing is a *tappability study* or a visual affordance test [4]. In these studies, designers have crowd workers or lab participants label interfaces for which elements they think are tappable and not tappable in a digital form or on paper. Based on this data, designers can construct heatmaps to help them visualize where users would tap in the app being tested. These studies can help designers discover which elements have missing or false tappability signifiers. However, in general, there is a lack of a dataset and deep understanding about interface tappability across diverse mobile apps. Having such a dataset and knowledge is required for us to create automated techniques to help designers diagnose tappability issues in their interfaces.

Crowdsourcing Data Collection

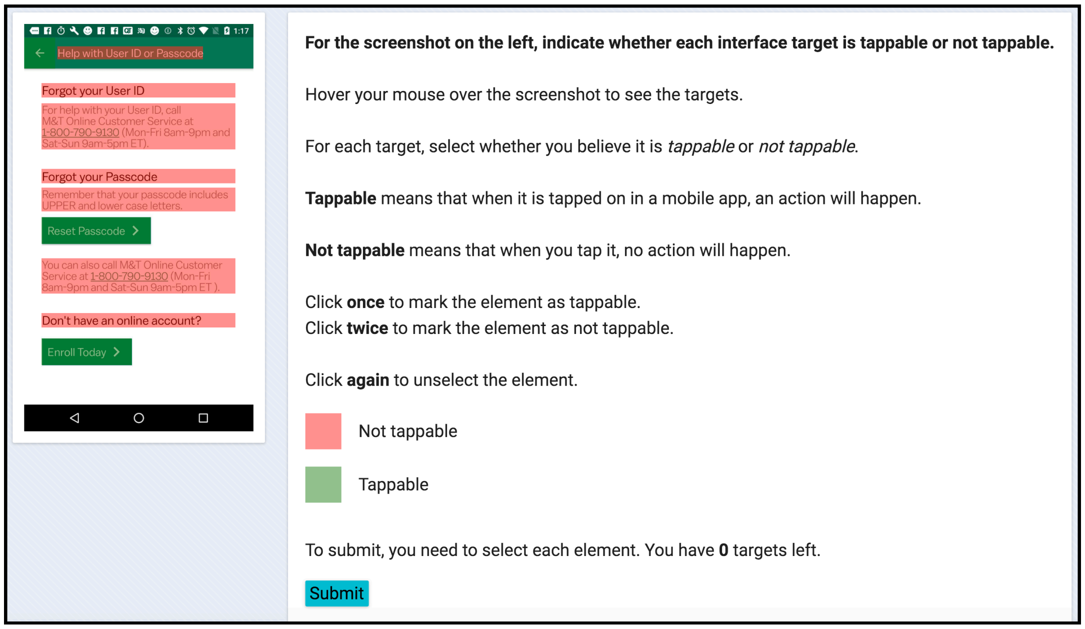
We designed a crowdsourcing task to simulate a tappability study across a large corpus of Android mobile apps [7]. We developed the worker interface as a web app (see Figure 2). The left side of the interface displayed a mobile app screenshot. The right side of the task interface displayed a set of instructions for the task, and an explanation about what we meant by tappable and not tappable. For tappable elements, this was *"When you tap this in a mobile interface, an action will happen."*, and for not tappable, the text was *"When you tap on it, no action will happen."*.

	Positive Class	#Elements	Precision	Recall
R1	`clickable=True`	6,878	89.97%	79.81%
	`clickable=False`	2,854	61.75%	78.56%
R2	`clickable=True`	7,423	90.02%	79.55%
	`clickable=False`	3,019	60.90%	78.30%
All	`clickable=True`	14,301	89.99%	79.67%
	`clickable=False`	5,873	61.31%	78.43%

Table 1: The total number of elements labeled in each round of the study, along the precision and recall of human workers in perceiving the actual clickable state of an element as specified in the view hierarchy metadata.

To collect our tappability dataset, we selected a set of 3,470 unique screens from a variety of mobile apps from the Rico dataset [7] and had crowd workers to label elements randomly sampled from these screens as either `tappable` or `not tappable`. We selected the elements for the workers to label in the following manner. Each UI screen in the Rico dataset has an Android view hierarchy that is a JSON tree structure of all of the interface elements on the screen (similar to a DOM tree for a web interface). Each element in the hierarchy has a `clickable` property that marks whether an element will respond to a tapping event—the intended behavior as specified by the app developer. For each screen, we selected up to five unique elements for both `clickable` and non-`clickable` ones. When selecting `clickable` elements, starting from a leaf element, we use the top-most `clickable` element in the hierarchy for labeling. When such an element is an inner node, it is often the case that the sub tree of elements are presented as a single interactive widget to the user which is more appropriate for the worker to label as a whole. We did not select any elements in the top status bar or bottom navigation bar which are standard across most screens in the dataset.

To perform a labeling task, a crowd worker hovers their mouse over the interface screenshot, and our web interface displays grey hotspots over the interface elements preselected based on the above process. Workers click on each hotspot to toggle the label as either tappable or not-tappable which are colored in green and red, respectively. We asked each worker to label around six elements for each screen. Depending on the complexity of a screen, the amount of elements could vary. We randomized the elements as well as the order to be labeled across each worker.


1:17

Help with User ID or Passcode

Forgot your User ID

For help with your User ID, call M&T Online Customer Service at 1-800-790-9130 (Mon-Fri 8am-9pm and Sat-Sun 9am-5pm ET).

Forgot your Passcode

Remember that your passcode includes UPPER and lower case letters.

Reset Passcode >

You can also call M&T Online Customer Service at 1-800-790-9130 (Mon-Fri 8am-9pm and Sat-Sun 9am-5pm ET).

Don't have an online account?

Enroll Today >

For the screenshot on the left, indicate whether each interface target is tappable or not tappable.

Hover your mouse over the screenshot to see the targets.

For each target, select whether you believe it is *tappable* or *not tappable*.

Tappable means that when it is tapped on in a mobile app, an action will happen.

Not tappable means that when you tap it, no action will happen.

Click **once** to mark the element as tappable.
Click **twice** to mark the element as not tappable.

Click **again** to unselect the element.

🟥 Not tappable

🟩 Tappable

To submit, you need to select each element. You have **0** targets left.

Submit

Figure 2: The crowdsourcing task interface we created to collect tappability labels.

Results

We collected 20,174 unique interface elements from 3,470 unique app screens. These elements were labeled by 743 unique workers in two rounds where each round involved different sets of workers. Each worker could complete up to 8 tasks. On average, each worker completed 4.67 tasks. Among these elements, 14,301 of them are indeed tappable, i.e., `clickable=True` in the view hierarchy, and 5,873 of them are not.

How well can human users perceive the actual clickable state of an element as coded by developers or designers? To answer this question, we treat the `clickable` value of an element in the view hierarchies as the real values and human labels as the predicted values for a precision and recall analysis. In this dataset of real mobile app screens, there were still many false signifiers for tappability that caused workers to misidentify tappable versus not-tappable elements (see Table 1). We found the workers were significantly more precise in labeling clickable elements than not-clickable ones. Even though misidentification for clickable elements was less common, workers still marked clickable elements as not tappable 20% of the time in this dataset. We found the results are quite consistent across two rounds of data collection studies involving different workers and interface screens. These results further confirmed that tappability is an important usability issue that is worth investigation.

Signifier Analysis

To understand how users perceive tappability, we conducted several analyses to reveal signifiers that are having an impact on tappability in real mobile apps. These findings have analytical value in understanding human perception on tappability and provide a basis for us to build machine learning models to predict tappability. We investigated several visual and non-visual features based on previous understandings of common visual signfiers [1, 2, 14] and through exploration of the characteristics of the dataset.

Element Type. There are conventions regarding how an element should appear for several element types, and because of this, users would consistently perceive them as tappable [23] (e.g., buttons). We examined how accurately workers label each type of interface element based on a subset of Android class types that are present in the Rico dataset [7] (see Figure 3). This figure shows the distribution of tappable versus not-tappable elements by type as labeled by human workers. Common tappable interface elements such as Button and Checkbox did appear more frequently in the set of elements marked as tappable.

Within each element type, we computed the accuracy by comparing the worker labels to the view hierarchy `clickable` values. For elements that were marked as tappable, the workers achieved high accuracy for most types. For not-tappable elements, the two most common types, TextView and ImageView, had accuracy percentages of only 67% and 45%, respectively. This is because these interface types allow for

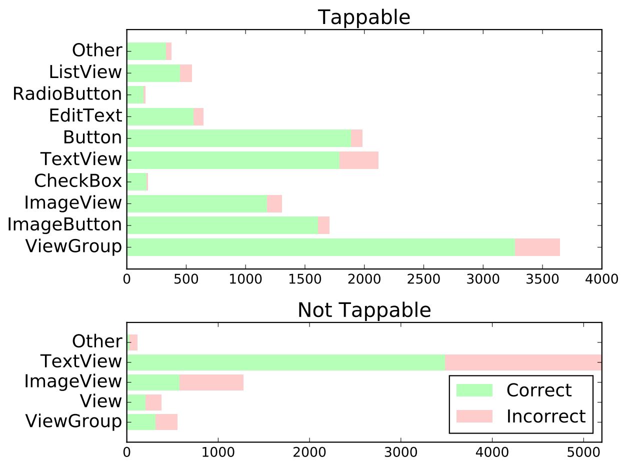


Figure 3: The the number of tappable and not-tappable elements within several type categories with the bars colored by the relative amounts of correct and incorrect labels in the type category.

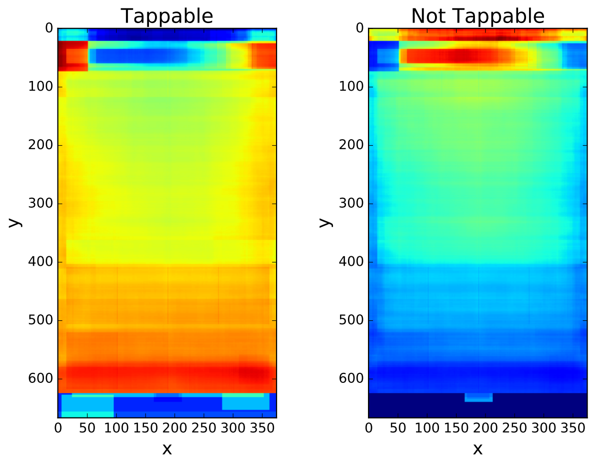


Figure 4: Heatmaps displaying the accuracy of tappable and not tappable elements by location. Workers guess higher percentages of not-tappable elements correctly towards the upper center of the interface, and tappable elements towards the bottom center of the interface.

more flexibility in design than standard element types such as RadioButton. Unconventional styles are more prone for ambiguity in tappability of an element.

Location. We hypothesize that the location of an element on the screen may also have influenced the workers to be less accurate at labeling its tappability. Figure 4 displays a heatmap representing the precision of the workers’ labels by location. We created the heatmap by computing the precision

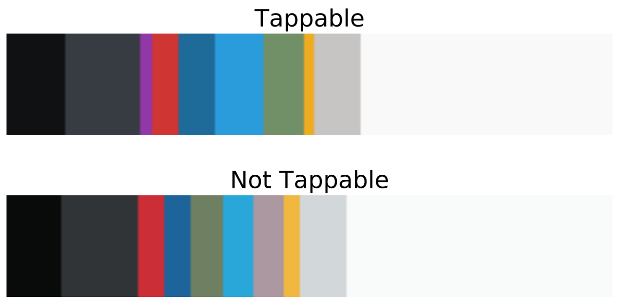


Figure 5: The aggregated RGB pixel colors of tappable and not-tappable elements clustered into the 10 most prominent colors found using K-Means clustering.

per pixel across all of the 20,174 elements we collected using the bounding box of each element from the Android view hierarchy. Warm colors represent higher accuracy values. When an element is identified as tappable, workers tend to be more accurate towards the bottom of the screen than the center top area. Placing any not-tappable element in these area might confuse people. There are two spots at the top region of high precision. We speculate that these spots are where apps tend to place their Back and Forward buttons. For not-tappable elements, the workers were less precise towards the bottom of the screen.

The workers achieved lower tappable precision in the area where the app header bar is typically located within a corresponding area of high precision for not tappable. This area is not tappable in many apps, so people may not realize any element placed there is tappable.

Size. There was only a small difference in average size between tappable and not-tappable elements across the dataset. However, for specific element types, size was a distinguishing factor. For ImageView elements, the average and median for not tappable elements were 2.39 and 3.58 times larger than those for tappable elements. People may believe larger ImageView elements, which typically display images, to be less likely to be tappable than smaller ImageView elements.

TextView elements usually display labels, but can also act as tappable elements. According to some design recommendations, it is best to give actionable elements, such as buttons, short actionable phrases that describe what will happen when the user performs the action. Labels should be short, plainly worded, and quick to read [29]. The text labels of not-tappable TextView element have an average and median of 1.48 and 1.55 times larger respectively than those of tappable TextView elements. This gives us a hint that TextView elements may be following these recommendations.

Color. Based on design recommendations [2], color can also be used to signify tappability. Figure 5 displays the top 10 most dominant colors among the set of worker-identified tappable and not-tappable elements, computed using a K-Means clustering algorithm. Looking at the brighter colors (e.g, blues, reds and greens), these colors have wider bars in the pixel clusters for tappable elements. Not-tappable elements have wider grey and white bars. We computed these clusters across the image pixels for 12 thousand tappable and 7 thousand not-tappable elements and scaled them by the proportion of elements in each set. Observing these differences shows us that color is likely a useful distinguishing factor.

4 TAPPABILITY PREDICTION MODEL

Because it is expensive and time consuming to conduct user studies, it is desirable to develop automated techniques to examine the tappability of mobile interfaces. Although it is possible to use the signifiers previously discussed as heuristics for this purpose, it would be difficult to manually combine them appropriately. It is also challenging to capture factors that are not obvious or hard to articulate. As such, we employed a deep learning approach to address the problem. Overall, our model is a feedforward neural network with a deep architecture (multiple hidden layers). It takes a concatenation of a range of features about the element and its screen and outputs a probability of how likely a human user would perceive an interface element as tappable.

Feature Encoding

Our model takes as input several features collected from both the view hierarchy metadata and the screenshot pixel data of an interface. For each element under examination, our features include 1) the semantics and functionality of the element, 2) the visual appearance of the element and the screen, and 3) the spatial context of the element on the screen.

Semantic Features. The label of an interface element can convey rich information about whether the element is interactive. For example, interface design literature recommends to label interactive elements with short, actionable phrases [29]. More frequently, non-interactive elements display information, or are labels for interactive elements. Length and text content are both potential tappability signifiers. For each element, we scan the text using OCR. To represent the semantics of the text, we use continuous word embedding that is a standard way of mapping words into a continuous vector that can be fed into a deep learning model. We encode the text label for each element as a 50-dimensional vector representation of words learned from a Wikipedia corpus [26]. When an element has multiple words in its label, we treat them as a bag of words and use the mean of their word embedding vectors as the semantic representation of the label.

Type Features. Interfaces are composed of many standard elements for which that users have learned the behaviors over time (e.g., buttons and checkboxes) [23]. In the Android platform, there are an array of interface element types that have evolved from standard elements such as buttons and checkboxes. However, new element types are frequently introduced such as the floating action button. In our model, we include the interface element type, an indicator of the element's functionality, as a feature to attempt to account for these learned conventions. We include a set of the 22 most common interface element types. These categorical values are represented as a continuous 6-dimensional embedding vector to be fed into the model. The embedding vectors are trainable parameters of the model.

Visual Features. As we discussed previously, visual design signifiers such as color distribution can help distinguish tappability of an interface element. In general, it is difficult to fully articulate the visual perception that might come into play and realize them as executable rules. As a result, we decide to feed the raw pixel values of an element as well as the screen to which the element belongs to the network, through convolutional layers—a popular approach for image processing. In particular, we resize the pixels of each element and format them as a 3D matrix in the shape of 32x32x3 where the height and width are 32 and 3 is the number of RGB channels. Contextual factors on the screen may also affect the perception of tappability of an interface element. Because of this, we resize and format the entire screen as another visual feature. This manifests as a 3D matrix in the shape of 184x300x3 and preserves the original aspect ratio. As we will discuss later, the contextual factors on the screen where an element is contain useful information for predicting the tappability of the element even though such information is not easy to articulate.

Spatial Features. Because the location and size are important signifiers of tappability, we include them as features as well. We capture the bounding box of the element as four scalar values: *x*, *y*, *width*, and *height*, and scale each value to the range of 0 and 1 by normalizing them using the screen width and height.

Model Architecture & Learning

Our model architecture is illustrated in Figure 6. To process the element and screenshot pixel data, our network consists of three convolutional layers with ReLU [21] as the activation function. Each convolutional layer applies a series of 8 3x3 filters to the image to help the model progressively create

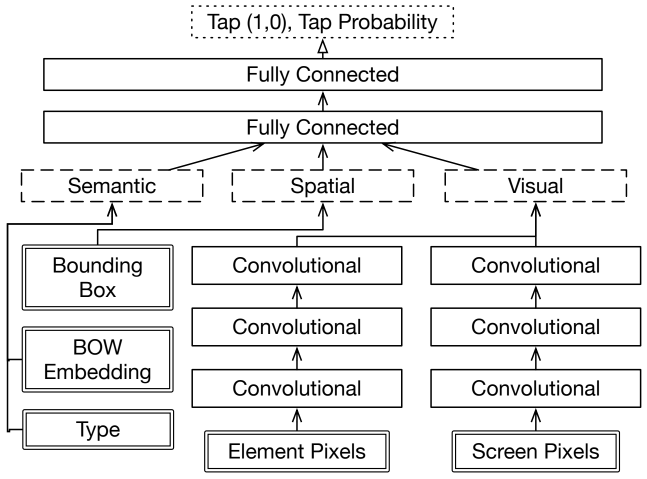


Figure 6: The tappability prediction model architecture

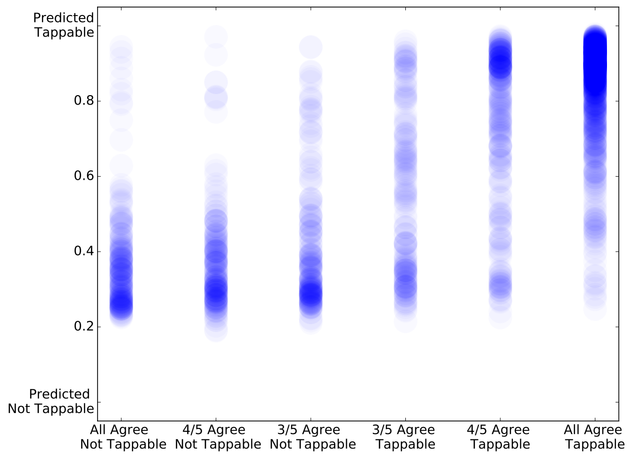


Figure 7: The scatterplot of the tappability probability output by the model (the Y axis) versus the consistency in the human worker labels (the X axis) for each element in the consistency dataset.

a feature map. In our network, each layer is followed by a 2x2 max pooling layer, which significantly reduces the dimensionality of the image data for processing. Finally the output of the image processing layers is concatenated with the rest of the features into a series of two fully connected dense layers (each layer is 100 dimensional), each again using ReLU [21] as the activation function.

The output layer produces a binary classification for the tappability of an element using a sigmoid activation function. The activation function transforms the output into probabilities in the range of zero to one. The probability indicates how likely the user would perceive the element as tappable in the context of its screen.

We trained the model by minimizing the sigmoid cross-entropy loss between the predicted values and the binary human labels on tappability of each element in the training data. For loss minimization, we used the Ada adaptive gradient descent optimizer with a learning rate of 0.01 and a batch size of 64. To avoid overfitting in the model, we applied a dropout ratio of 40% to each fully connected layer to regularize the learning. We implemented our model using the Tensorflow framework [5] in Python and the training was performed over a Tesla V100 GPU.

Model Performance Results

We evaluated our model using 10-fold cross validation based on the dataset we collected via crowdsourcing. In each fold, we used 90% of the data for training and 10% of the data for validation, and trained our model for 100,000 iterations. We used the value of 0.5 for thresholding the probability for tappable versus not tappable during validation. Overall, our model achieved validation precision and recall of 88.38% (SD: 1.07%) and 89.08% (SD: 1.73%) respectively, averaged across the 10 folds of the experiments. Note that the precision and recall measure both tappable and not-tappable cases. To understand what these performance measures imply, we analyzed how well the `clickable` attribute in the view hierarchy as specified by developers and designers can indicate user perception of tappability. The view hierarchy `clickable` attribute achieved a precision of 79.21% and recall of 79.31%. As a result, our model is significantly more accurate in approximating user perception about tappability which can help developers or designers identify mismatches on their mobile interfaces.

Human Consistency & Model Behaviors

We speculate that our model did not achieve even higher accuracy because human perception of tappability can be inherently inconsistent as people have their own experience in using and learning different sets of mobile apps. This can make it challenging for the model to achieve perfect accuracy. To examine our hypothesis, we collected another dataset via crowdsourcing using the same interface as shown in Figure 2. We selected 334 screens from the Rico dataset. These screens were not selected in our previous rounds of tappability data collection. We recruited 290 workers to perform the same task of marking each selected element as either tappable or not tappable. However, each element was labeled by 5 different workers for its tappability by which we intend to see how well these workers agree on the tappability of an element. In total, there were 2000 unique interface elements and each was labeled 5 times. In total, 1,163 elements (58%) were entirely consistent among all 5 workers which include both tappable and not-tappable elements. To analyze the consistency of the data statistically, we report two metrics. The first is in terms of an agreement score [30] which is computed using the following formula:

$$A = \frac{\sum_{e \in E} \sum_{r \in R} \left(\frac{|R_i|}{|R_e|} \right)^2}{|E|} \times 100\% \quad (1)$$

Here, e is an element in the set of all interface elements E that were rated by the workers, R_e is the set of ratings for an interface element e, and R_i is the set of ratings in a single category (0 - not tappable, 1 - tappable). We also report the consistency of the data using Fleiss' Kappa [10] which is a standard inter-rater reliability measure for measuring the agreement between a fixed number of raters assigning categorical ratings to a numbers of items. This measure is useful, in addition to the agreement score, because it computes the degree of agreement over what would be expected by chance. As there are only two categories, the agreement by chance is high. The overall agreement score across all elements using Equation 1 is *0.8343.* The number of raters is 5 for each element of a screenshot, and across 334 screenshots, resulting in an overall Fleiss' Kappa value of 0.520 *(SD=0.597, 95% CI [0.575,0.618], P=0).* This corresponds to a "Moderate" level agreement according to [17]. What these results demonstrate is that while there is a significant amount of consistency in the data, there still exists a certain level of disagreement on what elements are tappable versus not tappable. Particularly, consistency varies across element *type* categories. For example, *View* and *ImageView* elements were labeled far less consistently (*0.52, 0.63*) than commonplace tappable element types such as *Button* (94%), *Toolbar* (100%), and *CheckBox* (95%). View and ImageView elements have more flexibility in design, which may lead to more disagreement.

To understand how our model would predict elements that are inherently ambiguous on their tappability, we test our model that was previously trained from the cross-validation experiment on this new dataset. We found our model behavior matches the uncertainty in human perception of tappability surprisingly well (see Figure 7). When workers are consistent on the tappability of an element (two ends on the X axis), our model tends to give a more definite answer—a probability value close to 1 for tappable and close to 0 for not tappable. When workers are less consistent on an element (towards the middle of the X axis), our model is also less sure thus the probability is close to 0.5. We note that for predicting not-tappable elements, the probability of our model does not quite reach 0. We reason that this is because tappable and not-tappable elements in our training datasets are unbalanced (see Table 1)–there are more tappable elements than not-tappable ones in the dataset.

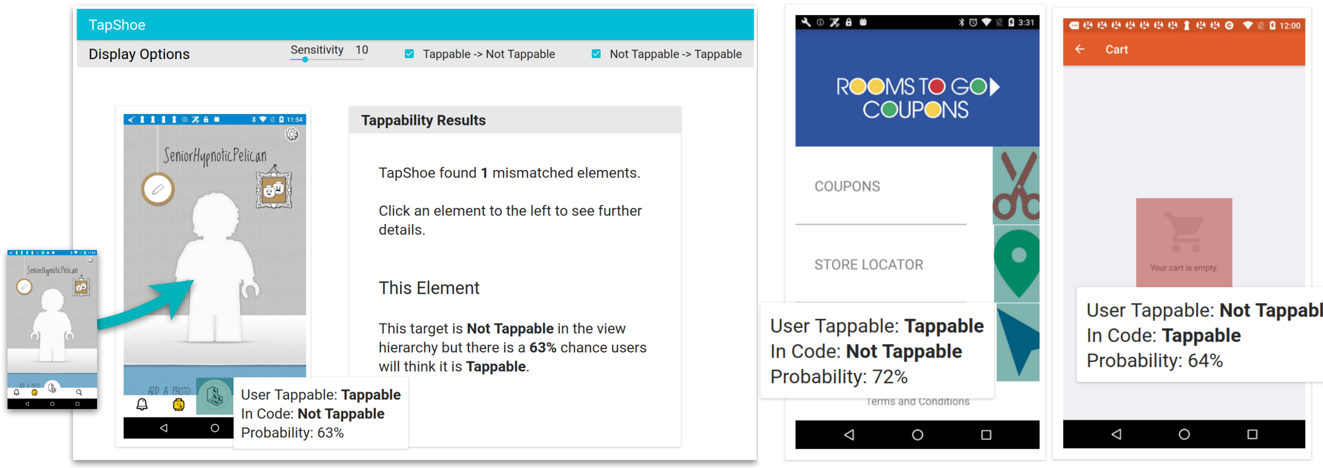
Screen Pixels Useful?

In addition, we wanted to understand how individual features contribute to the overall performance of our model. Particularly, one motivation to use deep learning is to alleviate the need for extensive feature engineering. Recall that we feed the entire screenshot of an interface to the model to capture contextual factors that might come into play but can not be easily articulated. We wanted to find out how well our model performs without the screenshot pixels as input. We found there is a noticeable drop of about 2% regarding both precision and recall: 86.62% (SD: 0.74%) and 86.22% (SD: 1.60%). This indicates that there is useful contextual information in the screenshot outside of the element being examined, which is affecting the users' decisions on tappability.

5 TAP SHOE INTERFACE

With the deep model we developed, we created a web interface for TapShoe (see Figure 8). The interface can help mobile app designers and developers examine the tappability of their interfaces. We describe the TapShoe interface from the perspective of an app designer, Zoey, who is designing an app for deal shopping, shown in right hand side of Figure 8. Zoey has recently redesigned some icons to be more colorful on the home page links of her app for "Coupons", "Store Locator", and "Shopping". Zoey wants to understand how the changes she has made would affect the users' perception of which elements in her app are tappable. First, Zoey uploads a screenshot and view hierarchy for her app by dragging and dropping them into the left hand side of the TapShoe interface. Once Zoey drops her screenshot and view hierarchy, TapShoe analyzes her interface elements, and returns a tappable or not-tappable prediction and probability for each element. The TapShoe interface displays highlights over the interface elements where the the actual tappable state of each element in the view hierarchy does not match up with the users' perception.

Zoey sees that the TapShoe interface highlighted the three colorful icons she redesigned. These icons were not tappable in her app but TapShoe predicted that the users would perceive them as tappable. She examines the probability scores for each element by clicking on the green hotspots on the screenshot to see informational tooltips. She adjusts the sensitivity slider to change the threshold for the model's prediction. Now, she sees that the "Coupons" and "Store Locator" icon are not highlighted and that the arrow icon has the highest probability of being perceived as tappable. In the end, she decides to make all three colorful icon elements interactive and extend the tappable area next to "Coupons", "Store Locator", and "Website", thus preventing her users from the frustration of tapping on these elements with no response.



Tappability Results

TapShoe found **1** mismatched elements.

Click an element to the left to see further details.

This Element

This target is **Not Tappable** in the view hierarchy but there is a **63%** chance users will think it is **Tappable**.

User Tappable: **Tappable**
In Code: **Not Tappable**
Probability: 63%

User Tappable: **Tappable**
In Code: **Not Tappable**
Probability: 72%

User Tappable: **Not Tappable**
In Code: **Tappable**
Probability: 64%

Figure 8: The TapShoe interface. An app designer drags and drops a screenshot and view hierarchy on the left, and TapShoe highlights the interface elements where the predicted user tappable state is different than the actual tappable state of the element in code.

We implemented the TapShoe interface as a JavaScript web application, with a Python web server. The web server process an image and JSON view hierarchy. The web server traverses the view hierarchy, and returns all the interface elements from the view hierarchy. The web server queries a trained model, hosted through the Tensorflow model serving API, to retrieve the predictions for each elements and returns them to the web client. The model server is hosted through a Docker container.

6 INFORMAL FEEDBACK FROM DESIGNERS

To understand more about whether TapShoe would be useful in a real design context, we conducted informal design walkthroughs with 7 professional interface designers at a large technology company. The designers worked on design teams for three different products. First, we asked them to describe how they have used tappability studies to uncover potential problems with their designs. We demonstrated the TapShoe interface to them and collected informal feedback on the idea of getting tappability predictions from the TapShoe model, and on the TapShoe interface for helping app designers identify tappability mismatches. We also asked them to envision new ways they could use the tappability prediction model beyond the functionality of the TapShoe interface.

The designers responded positively to the use of TapShoe as a way to uncover usability issues on tappability. They gave several directions to improve the tool. Particularly, the following themes have emerged.

Visualizing Probabilities

The designers saw high potential in being able to get tappability probability score for the elements in their design. Currently, the TapShoe interface displays only the probabilities for the elements that have a mismatch based on the threshold set by the sensitivity slider in the tool. However, several of the designers mentioned that they would want to see the probability scores for all elements. This could allow them to get a quick glance at the tappability of their designs as a whole. Presenting this information in a heatmap that adjusts the colors based on the tappability scores could help them visually compare the relative level of tappability of each element. This would allow them to deeply examine and compare interface elements for which tappability signifiers are having an impact. They also mentioned that sometimes, they do not necessarily aim for tappability to be completely binary. Tappability could be aimed to be higher or lower along a continuous scale depending on the importance of an element. In an interface with a primary action and secondary action, they would be more concerned that people perceive the primary action as tappable than the secondary action.

Exploring Variations

The designers also saw the potential of the TapShoe model for helping them systematically explore variations. TapShoe's interface only allows a designer to upload a single screen, however, the designers envisioned an interface to upload and compare multiple versions of their designs to systematically change signifiers and see how that changes the model's prediction. This could help them discover new design principles to make interface elements look more or less tappable.

It could also help them compare more granular changes at an element level, such as different versions of a button design. As context within a design can also affect an element's tappability, they would want to move elements around and change contextual design attributes to have a more thorough understanding of how context affects tappability. Currently, the only way for them to have this information is to conduct a large tappability study which limits them to trying out only a few design changes at a time. Having the TapShoe model output could greatly expand their current capabilities for exploring design changes that may affect tappability.

Model Extension and Accuracy

Several designers also wondered whether the model could extend to other platforms. In some cases, they design desktop or web interfaces that could benefit from this type of model. Additionally, they have collected data that our model could already use for training. We believe our model could help them in this case as it would be simple to extend to other platforms or to use existing tappability data for training.

We also asked the designers about the importance of the accuracy of the model. However, the designers already thought that the model could be useful in its current state even for helping them understand the relative tappability of different elements. Providing a confidence interval for the prediction could aid in giving them some amount of trust in the prediction.

7 DISCUSSION

Our model achieves good accuracy at predicting tappable and not-tappable interface elements and the TapShoe tool is well received by designers who thought TapShoe is already a useful tool for their real design projects. Here we discuss the limitations and elaborate on directions for future work. One limitation of our model is that we have only trained it on the dataset from a specific interface platform (i.e. Android) and therefore the results potentially may not generalize to other platforms. However, our model relies on general features that are available across many interface platforms (e.g., element bounding boxes and types). Additionally, we collected a large amount of tappability annotation labels for a relatively small cost using crowdsourcing, so it would not be infeasible for mobile platform providers or companies to collect a similar dataset for training the model.

Additionally, tappability signifiers can change over time as new design concepts are introduced. Thus our model may need to be trained with additional data if these design concepts were drastically different. However, most design changes are not drastic enough that our model could not already account for them and the cost and time for re-training would be small.

Through our consistency evaluation, we came to understand that people's understandings of tappability is not always consistent. In the future, we plan to explore ways to improve the models behavior with inconsistent data. We believe that there are many opportunities for improvement, both on making our model more powerful, and in using the predictive power of machine learning models to automate different aspects of usability evaluation.

Future work could focus on identifying features that would add predictive power to the model. The features we identified are a good starting point, but there may be potentially many other features that might affect the users decision. Additionally, identifying and capturing more of the potential reasons behind a users tappable or not tappable perception could enable us to provide a deeper recommendation for a suggested fix.

To offer a suggestion for fix, beyond identifying a tappability issue, we need to identify the reasons why or the signifiers behind the a user's perception of tappability and communicate them with the designer in a humanly understandable way. There are two approaches to pursue this. One is to analyze how the model relies on each set of features, although understanding the behavior of a deep learning model is generally challenging and it is an active area in the deep learning field. The other approach is to train the model to recognize the human reasoning behind their selection. The progress in this direction will allow a tool to provide a more complete and useful output to the designers. Additionally, as we begin to understand more of the features that people use to determine which elements are tappable and not tappable, we can easily incorporate these new features in a deep learning model as long as they are manifested in the data.

8 CONCLUSIONS

We present an approach for modeling and predicting tappability of mobile interface elements. We collected a large scale dataset via multiple rounds of crowdsourcing studies, conducted an in-depth analysis about tappability, and designed and trained a deep model that uses a rich set of features that can affect human perception of tappability. Our model achieved reasonable accuracy in approximating human perception on tappability, and we also conducted an analysis into the behavior of the model in relation to uncertainty in human perception on tappability. Lastly, we developed the TapShoe interface that uses the trained deep model to automatically examine a mobile interface based on its screenshot and UI description. TapShoe received positive feedback from 7 professional interaction designers who thought TapShoe would be a useful tool for their realistic design projects.

REFERENCES

[1] [n. d.]. Material Design Guidelines. ([n. d.]). https://material.io/design/

[2] 2015. Beyond Blue Links: Making Clickable Elements Recognizable. (2015). https://www.nngroup.com/articles/clickable-elements/

[3] 2017. Flat UI Elements Attract Less Attention and Cause Uncertainty. (Sep 2017). https://www.nngroup.com/articles/flat-ui-less-attention-cause-uncertainty/

[4] 2018. Visual Affordance Testing. (Sep 2018). http://practicaluxmethods.com/product/visual-affordance-testing/

[5] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.

[6] Zoya Bylinskii, Nam Wook Kim, Peter O'Donovan, Sami Alsheikh, Spandan Madan, Hanspeter Pfister, Fredo Durand, Bryan Russell, and Aaron Hertzmann. 2017. Learning Visual Importance for Graphic Designs and Data Visualizations. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 57–69. https://doi.org/10.1145/3126594.3126653

[7] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 845–854. https://doi.org/10.1145/3126594.3126651

[8] Biplab Deka, Zifeng Huang, Chad Franzen, Jeffrey Nichols, Yang Li, and Ranjitha Kumar. 2017. ZIPT: Zero-Integration Performance Testing of Mobile App Designs. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 727–736. https://doi.org/10.1145/3126594.3126647

[9] Biplab Deka, Zifeng Huang, and Ranjitha Kumar. 2016. ERICA: Interaction Mining Mobile Apps. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 767–776. https://doi.org/10.1145/2984511.2984581

[10] Joseph L Fleiss. 1971. Measuring Nominal Scale Agreement Among Many Raters. *Psychological bulletin* 76, 5 (1971), 378.

[11] William W. Gaver. 1991. Technology Affordances. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*. ACM, New York, NY, USA, 79–84. https://doi.org/10.1145/108844.108856

[12] James J Gibson. 1978. The Ecological Approach to the Visual Perception of Pictures. *Leonardo* 11, 3 (1978), 227–235.

[13] Michael D. Greenberg, Matthew W. Easterday, and Elizabeth M. Gerber. 2015. Critiki: A Scaffolded Approach to Gathering Design Feedback from Paid Crowdworkers. In *Proceedings of the 2015 ACM SIGCHI Conference on Creativity and Cognition (C&#38;C '15)*. ACM, New York, NY, USA, 235–244. https://doi.org/10.1145/2757226.2757249

[14] IDF Instructor. [n. d.]. Affordances and Design. ([n. d.]). https://www.interaction-design.org/literature/article/affordances-and-design

[15] Aniket Kittur, Ed H. Chi, and Bongwon Suh. 2008. Crowdsourcing User Studies with Mechanical Turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 453–456. https://doi.org/10.1145/1357054.1357127

[16] Steven Komarov, Katharina Reinecke, and Krzysztof Z. Gajos. 2013. Crowdsourcing Performance Evaluations of User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 207–216. https://doi.org/10.1145/2470654.2470684

[17] J Richard Landis and Gary G Koch. 1977. An Application of Hierarchical Kappa-type Statistics in the Assessment of Majority Agreement Among Multiple Observers. *Biometrics* (1977), 363–374.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *nature* 521, 7553 (2015), 436.

[19] Yang Li, Samy Bengio, and Gilles Bailly. 2018. Predicting Human Performance in Vertical Menu Selection Using Deep Learning. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 29.

[20] Kurt Luther, Jari-Lee Tolentino, Wei Wu, Amy Pavel, Brian P. Bailey, Maneesh Agrawala, Björn Hartmann, and Steven P. Dow. 2015. Structuring, Aggregating, and Evaluating Crowdsourced Design Critique. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '15)*. ACM, New York, NY, USA, 473–485. https://doi.org/10.1145/2675133.2675283

[21] Vinod Nair and Geoffrey E Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th international conference on machine learning (ICML)*. 807–814.

[22] Michael Nebeling, Maximilian Speicher, and Moira C Norrie. 2013. CrowdStudy: General Toolkit for Crowdsourced Evaluation of Web Interfaces. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering Interactive Computing Systems*. ACM, 255–264.

[23] Don Norman. 2013. *The Design of Everyday Things: Revised and Expanded Edition.* Constellation.

[24] Donald A. Norman. 1999. Affordance, Conventions, and Design. *interactions* 6, 3 (May 1999), 38–43. https://doi.org/10.1145/301153.301168

[25] Donald A. Norman. 2008. The Way I See It: Signifiers, Not Affordances. *Interactions* 15, 6 (Nov. 2008), 18–19. https://doi.org/10.1145/1409040.1409044

[26] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[27] Ken Pfeuffer and Yang Li. 2018. Analysis and Modeling of Grid Performance on Touchscreen Mobile Devices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 288, 12 pages. https://doi.org/10.1145/3173574.3173862

[28] Hanna Schneider, Katharina Frison, Julie Wagner, and Andras Butz. 2016. CrowdUX: A Case for Using Widespread and Lightweight Tools in the Quest for UX. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems (DIS '16)*. ACM, New York, NY, USA, 415–426. https://doi.org/10.1145/2901790.2901814

[29] Jenifer Tidwell. 2010. *Designing interfaces: Patterns for effective interaction design.* " O'Reilly Media, Inc.".

[30] Jacob O Wobbrock, Htet Htet Aung, Brandon Rothrock, and Brad A Myers. 2005. Maximizing the Guessability of Symbolic Input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Extended Abstracts*. ACM, 1869–1872.

[31] Anbang Xu, Shih-Wen Huang, and Brian Bailey. 2014. Voyant: Generating Structured Feedback on Visual Designs Using a Crowd of Non-Experts. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, 1433–1444.