5-2018

Clarity: An Exploration of Semantic Information Encoded in Mobile Application GUIs

Michael J. Curcio
College of William and Mary

Clarity: An Exploration of Semantic Information Encoded in Mobile Application GUIs

Michael J. Curcio Jr.

Williamsburg, VA

A Thesis presented to the Faculty
of the College of William and Mary in Candidacy for the Degree of
Bachelors of Science

Department of Computer Science

The College of William and Mary
May 2018

APPROVAL PAGE

This thesis is submitted in partial fulfillment of
the requirements for the degree of

Bachelors in Computer Science

[illegible]

Michael J. Curcio Jr.

Approved by the Committee, May 2018

[illegible]

Committee Chair
Associate Professor Denys Poshyvanyk, Computer Science
The College of William and Mary

Robert Michael Lewis

Associate Professor Robert M. Lewis, Computer Science
The College of William and Mary

[illegible]

Associate Professor Peter Kemper, Computer Science
The College of William and Mary

[illegible]

Associate Professor Ross Iaci, Mathematics
The College of William and Mary

ABSTRACT

Upon installing a mobile application, human beings are able, to a great extent, to know immediately what the subcomponents of the screen do. They know what buttons return them to the previous screen, which ones submit their log in information, and which brings up the menu. This is the result of a combination of intuitive design and cross-platform design standards which allow users to draw on previous experience. Regardless, the fact that humans are able to understand the functionality of screen components at a glance suggests that there is *semantic information* encode into a mobile application's GUI.

In this work, we present an automated approach to exploring the nature of the semantic information encoded into the GUI of a mobile application. We do this using three modalities (1) a screenshot of an image, (2) text descriptions of the functionality of GUI components sourced through Amazon's Mechanical Turk, and (3) parsed information from the screen hierarchy's XML dump. The first two modalities are aligned using a convolutional neural network, which detects objects in the screenshot and extracts salient features, paired with a bidirectional recurrent neural network which serves as a language model. Both of these models maps their respective modalities to a *semantic space*, and then aligns the two representations in that space. The third modality is incorporated by using a Seq2Seq model which maps the screen's XML dump directly to reasonable descriptions of the functionality of the screen.

Our experiments reveal that semantic information extracted from the above representations of the GUI of a mobile application is comprable to that of real-world images such as those found in the MSCOCO dataset. In this work, we compare our results to similar models trained on this dataset, and compare the results from different screen representations against eachother.

TABLE OF CONTENTS

List of Tables iii

List of Figures iv

1 Introduction 2

2 Background & Related Work 7

2.1 Deep Learning Background 7

2.2 Properties of Mobile Application GUIs 8

2.2.1 The Screen Hierarchy 8

2.2.2 UIAutomator 9

2.3 Image Classification and Object Detection 10

2.4 Deep Visual-Semantic Embeddings 10

2.5 Recurrent Neural Network Language Models 11

2.6 Image and Region Captioning 11

3 Approach 13

3.1 Data-Collection 14

3.2 Image Representations 15

3.3 Sentence Representation 17

3.3.1 Pre-processing 17

3.3.2 The Model 17

3.3.3 Mapping to the Semantic Space 18

3.4 Aligning Modalities 18

3.5 Hyperparameter Tuning 19

3.6 Hierarchy Representation 20

3.6.1 The Model 21

3.6.2 Implementation 22

3.6.3 Motivation Behind Seq2Seq 23

4 Experimental Design 24

4.1 Quantitative Evaluation 25

4.1.1 BLEU Score 25

4.1.1.1 The N-Gram Computation 25

4.1.1.2 Implementation 26

4.2 RQ_1: Comparison to real-world semantics 26

4.3 RQ_2: Comparison Between Screen Representations 26

5 Results 28

5.1 RQ_1 28

5.2 RQ_2 30

6 Conclusion 32

6.1 Limitations and Avenues for Improvement 32

6.2 Future Work 33

LIST OF TABLES

LIST OF FIGURES

1.1 Conceptual image for ideal output of Clarity. Note that the detected objects are described both in their locations on the screen and also the functionality of the component found within 4

2.1 Simple example of how the screen hierarchy works. Note that components that are subsumed by containers are under them in the tree. A component can only be contained in one container, they cannot overlap across multiple containers. The only *leaf level component* listed here is the ImageButton. 9

3.1 How objects are detected within the image and then represented as a set of vectors. 16

3.2 Hyper parameter values used for final model 21

5.1 Comparison of Clarity with image-captioning frameworks for real-world objects. MLBL-B-Conv refers to a Modality-Based Log Bilinear Model with a convolutional network, and LBL is a baseline Log Bilinear Model 29

5.2 Example output of clarity. Note that spatial references are given. We chose these examples to give a representative snapshot of the capabilities and shortcomings of clarity. 30

5.3 Comparison of ReDraw with the Seq2Seq model which uses as input the parsed UI XML dumps. 31

Clarity: An Exploration of Semantic Information Encoded in Mobile Application GUIs

Chapter 1

Introduction

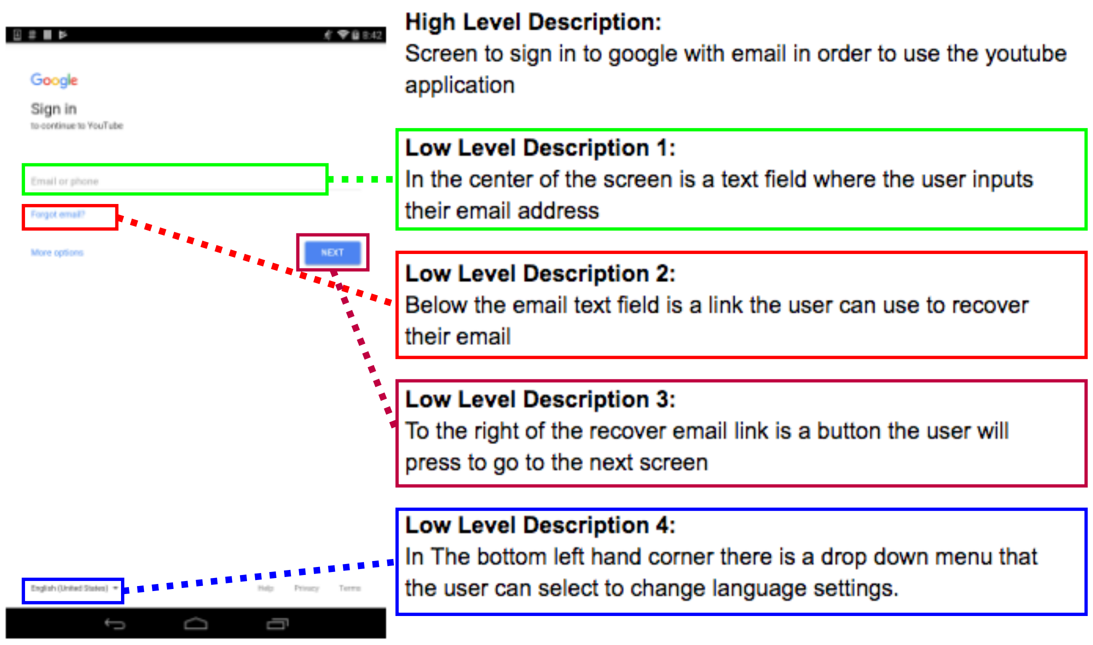
It is quite remarkable that, from the moment someone unlocks their mobile device, they have an instinctive understanding of how to navigate the piece of technology. They have some inkling of what will happen if they click this button, or what setting that slider controls. Moreover, effective mobile application design is predicated on the fact that, given a well-design application, users will intuitively understand how to manipulate it, to a large extent extent, immediately upon interacting with it for the first time.

No doubt, much of the above can be attributed to effectual design principles for mobile graphical user interfaces (GUIs), as well as the standardization of design across platforms and applications. Much work has been done to optimize how humans interact with computers through graphical interfaces [35], and similar work has been done that is specific to mobile devices [13]. The popularization of such design principles and practices has meant that regardless of platform or application, users can draw on previous experiences to help them learn the interface. This is partially due to the fact that a well-known approach to designing *human-computer interaction* (HCI) is the cognitive approach, which seeks to take advantage of aspects of applied psychology to build intuitive GUIs [8, 11]. Moreover, this goal is often explicitly sought after in modern contexts, with cross-platform GUI frameworks such as *React Native* and *Xamarin*. However, the standardization of these design principles also means that the functionality of the application represented by the screen can be reasonably inferred just by looking at it. In other words, semantic information about the application is embedded into the screenshot of the screen. We seek to explore the nature of this semantic information in this work.

We frame the task of extracting semantic information from the screenshot of a mobile application as an image-region captioning problem, much like Karpathy et. al [19]. . Image captioning is a well-studied problem, and many approaches have been proposed to faithfully capture the nature of an image in a short natural language sentence [38, 30, 17, 14]. To support this endeavor, a multitude of datasets have been developed to support supervised training for deep models [23, 32]. At a superficial glance, our work may be interpreted as an extension of these approaches to the domain of software engineering, and mobile application user interfaces more specifically. That is, while the aforementioned work is focused on captioning or describing images generally, we focus on using these techniques to describe GUI functionality in natural language, taking just a screenshot as input. However, there are unique aspects regarding our problem domain that dramatically set our work apart and illustrate the potential future impact on research in SE.

Our problem domain departs significantly from the general image-captioning problem. We do not seek to simply describe what is on the screen in natural language, rather, we seek to describe, not only the screen components and their locations, but what the *functionality* or *purpose* of these components are. That is, we want create a fully automated approach for forming a deeper understanding of mobile application GUIs. The primary goal of this work can be summarized in the question, "*What is the nature of the semantic information encoded into mobile application GUIs?*" As will be revealed below, this necessitates an entirely different approach to labelling data for supervised learning. To this point, supervised learning approaches to the image-captioning problem focus on *what* is present; we take the next step and ask: *why* are these things present?

The ability to automatically describe the intended functionality of screen components in a mobile application GUI is an important first step with regards to existing problems in software engineering. Take, for example, automated documentation production, such



High Level Description:
Screen to sign in to google with email in order to use the youtube application

Low Level Description 1:
In the center of the screen is a text field where the user inputs their email address

Low Level Description 2:
Below the email text field is a link the user can use to recover their email

Low Level Description 3:
To the right of the recover email link is a button the user will press to go to the next screen

Low Level Description 4:
In The bottom left hand corner there is a drop down menu that the user can select to change language settings.

Figure 1.1: Conceptual image for ideal output of Clarity. Note that the detected objects are described both in their locations on the screen and also the functionality of the component found within

as the automated description of Java classes found in Moreno et. al [29]. It is impossible to produce useful documentation for GUI code without knowing the *purpose* of the GUI components themselves. Demonstrating that this information is (1) encoded into mobile application GUIs, and (2) that this semantic information can be learned, speaks to the feasibility of learning approaches to a wide range of topics which are dependent on understanding the functionality of the application or piece of software in question. These topics include but are not limited to the aforementioned *documentation production*, *automated summarization of app execution*, and ultimately the much sought-after *program synthesis*. While in this work, we use the semantic information encoded in GUIs to simply describe them with respect to their functionality, this semantic information can also be harnessed to make progress in the three aforementioned topics, and in a wide range of others not mentioned.

Concretely, the contributions of this work can be summarized in the following points:

1. **Dataset** - We have constructed a completely new dataset from scratch consisting of

roughly 14,000 image-description pairings. This dataset was filtered by the authors of this paper, and sourced through Amazon's Mechanical Turk. An image-description pair consists of a screenshot of a running mobile application (the image) and 5 natural language descriptions of the image: one "high level description", and four "low level description". This dataset will be made open to the public upon publication of this paper.

2. **Multi-Modal Model for Captioning Images** - Based heavily on the *neuraltalk2* tool developed by Karpathy et. al [19], we adopt this approach for our problem domain and produce a pipeline which takes a screenshot of a running mobile application as input and outputs descriptions of the *functionality* of regions of the screenshot (i.e. individual screen components or small groups of components). That is, the model is the tool that we use to extract semantic information from the image representation of the GUI. We implement this approach in a framework we call **Clarity**. This tool consists of a convolutional neural network with a VGGNet architecture [36] and a bi-directional recurrent neural network [34] which serves as the language model. Code and models trained on the dataset above will be released upon publication of this paper.

3. **Seq2Seq Model** - In addition to Clarity, we present a Seq2Seq model which maps **GUI metadata** directly to target sequences. The GUI metadata is obtained by parsing the XML dump of the screen layout hierarchy, and extracting varying properties of the componets therein. In total, we have 4 different configurations for extracting sequences from the metadata. The model is then trained on metadata-description pairs, each consisting of the sequence extracted from the metadata using the tool uiautomator, and the target sequences obtained from the original dataset. The implementation and trained models are to be released on publication of this paper.

4. **Comprehensive Evaluation** - We perform an evaluation of Clarity and the Seq2Seq model, and compare these results against one another to determine if different GUI representations

lead to different extracted semantics. We also briefly compare semantics extracted by Clarity to state-of-the-art image-to-description frameworks.

Chapter 2

Related Works

In this work we investigate whether the semantic information encoded into images depicting the GUI of a mobile application can be used to infer functional attributes of the underlying program. To accomplish this, we leverage recent advances in the fields of machine learning (more specifically deep learning), computer vision, and pattern recognition. Thus, in this section we provide background on Deep Learning approaches for machine learning and discuss relevant related approaches that aim to caption images with natural language descriptions.

2.1 Deep Learning Background

We begin with a brief overview of deep learning and some of the techniques therein, to give the unfamiliar reader a foothold in the material we will discuss below.

In general, there are two categories of deep learning techniques, (1) *supervised learning* and (2) *unsupervised learning.* All techniques discussed in this paper will be examples of the former. In a supervised learning problem, we have a set of inputs X and a set of desired outputs (also called targets, or ground truths) Y. Often, X is seen as an $m \times n$ matrix, with m being the number of training instances (size of the training dataset) and n being the number of input features in each training instance. Y is often seen as an m-dimensional vector, with Y_i being the desired output corresponding to the features in the i^{th} row of the input matrix. The goal is to find a mapping $f : X \to Y$ which maps inputs to their desired outputs.

More concretely, during training, we must choose a *loss function* which measures the difference between the desired output, y_i, and the actual output $f(x_i)$, with f some mapping between X and Y. Denote this function $L(f(x_i), y_i)$. Denote f^* our "ideal" function, the one that minimizes the expected value of the loss. Owing to the non-completeness of the input space, analytical methods cannot work without making very strong (and unfounded) assumptions about the dataset. However, if we assume that the dataset is independently and identically distributed (often shortened as i.i.d) we can approximate the $f*$ as follows

$$f* = \arg\ \min \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i)$$

The interested reader can find a more complete summarization of deep learning techniques in Section 2 Andrej Karpathy's dissertaion [18]

2.2 Properties of Mobile Application GUIs

Throughout this work, we will discuss aspects of mobile application GUIs and their structure. For the convenience of the reader, we outline some of the most important aspects below.

2.2.1 The Screen Hierarchy

Modern GUIs for mobile applications are constructed in a tree structure, and this tree structure we refer to as the *hierarchy.* In order to discuss them generally, we partition all types of screen components (called *widgets* in official Android documentation) into two classes: (1) *containers* and (2) *leaf-level components.* Containers are, as the name suggests, components which serve to hold child components, and organize them on the screen in a desired way. For instance, Android's *LinearLayout* aligns its child components either

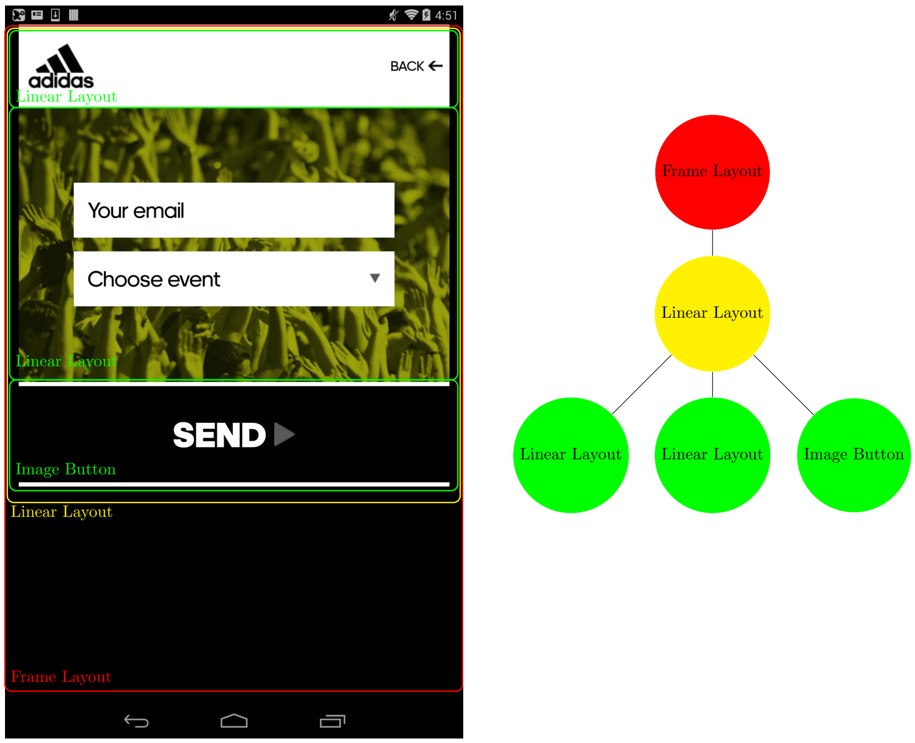


Figure 2.1: Simple example of how the screen hierarchy works. Note that components that are subsumed by containers are under them in the tree. A component can only be contained in one container, they cannot overlap across multiple containers. The only *leaf level component* listed here is the ImageButton.

vertically or horizontally, and inside the bounding box specified by the layout component. Leaf-level components are components that cannot have any children, and that typically are rendered in a GUI. Android components fitting this description are, for instance, *ImageButton*, *TextView*, and *NumberPicker*.

2.2.2 UIAutomator

In order to make use of the semantic information encoded into the layout hierarchy, discussed above, we must be able to obtain it at runtime. To accomplish this, we make use of a tool called *uiautomatorviewer*. This is a dynamic analysis tool that displays the runtime hierarchy of an application. The screen hierarchy is displayed as a tree, and for each component we are able to extract properties such as its bounding box, text, component type, and whether it is a clickable component.

2.3 Image Classification and Object Detection

Convolutional Neural Networks (CNNs) have emerged as a powerful tool both for the classification of images and for object detection [21, 42, 36]. This supervised learning technique is able to learn robust features on benchmark data sets such as ILSVRC [33]. Even shallower, more light-weight networks with less parameters, such as AlexNet [21], show strong results on these datasets. Deeper networks were then developed such as ResNet [15], GoogLeNet [40], and VGGNet [36], the last of which boasts a 23.7% top-1 error, and a 6.8% top-5 error. Moreover, Moran et. al. [28] have illustrated that CNNs also show great promise in recognizing and classifying mobile application UI components, achieving a top-1 accuracy of 91% with a relatively shallow network architecture.

2.4 Deep Visual-Semantic Embeddings

The approach in this work is inspired partly by Frome et. al [9], aimed at overcoming two shortfalls of CNNs for visual recognition systems: they are limited to a discrete number of categories for image classification, and CNNs treat each classification category as semantically unrelated. In addition to achieving equivalent top-10 precision with AlexNet [21], the visual-semantic embedding model is more likely to predict semantically similar categories to the ground truth, and can generalize to categories outside the model's training set, i.e. *zero-shot learning*. Preceding this work was Socher et. al. [37], who trained a deep neural model (unsupervised) and a neural language model in tandem, to obtain image representations and embedding representations for thousands of words respectively. They then learned a linear mapping between the two spaces to link the image and word representations. Barnard et. al. [2] also produced an eclectic study of the relationship between images and natural language descriptions. For our work, we focus on learning the semantics of particular regions of an image, and in this way it closely mirrors [19], to be discussed below.

2.5 Recurrent Neural Network Language Models

Recurrent Neural Networks (RNNs) have great expressive capabilities, and have been used for tasks such as general text generation [39]. Mikolov et. al [25] showed that RNNs can be used to learn semantically-meaningful, dense vector representations of words (i.e. words that are semantically similar will have similar vector representations), and this learning can be done in a reasonable amount of time on very large datasets. RNNs are able to consider significantly more context when making predictions than feedforward models [26], which only consider k previous words, with k being fixed [3]. Employing long short term memory units (LSTMs) can significantly improve the ability of the RNN to utilize context from many time-steps previous [16] at the cost of increased training time [19].

2.6 Image and Region Captioning

Describing an image with a sentence (as opposed to giving it a label) is a well known problem, and there have been a variety of approaches taken thus far. Some have chosen to frame it as a ranking problem, employing techniques resembling K-Nearest-Neighbors (KNN) [38, 30, 17], where test images are given the description of the "nearest" image in the training set. Representations for images are typically learned by a CNN in this case. Others employ fixed templates which are meant to serve as the semantic space between image and description [17, 14]. These sorts of solutions restrict the space of generated descriptions and are not readily applied to captioning regions. Kiros et. al. [20] departs from these methods by using a log bilinear model (LBL) to generate sentence descriptions. In the case of generating sentence descriptions for images, they condition the LBL on the feature representation of an image obtained by a forward pass through a CNN. However, the LDL model has a fixed context size, limiting the flexibility of the language model. To overcome this, and most closely related to our approach, Karpathy et. al. [19] replaces the LBL model with a *simple* RNN language Model (RNNLM). This RNNLM is simple in that it does not make use of LSTM units, which reduces training time. The use of an RNNL as opposed to LBL model allows variable context for each word generated, producing more meaningful word embeddings. Our approach differs from this work in two ways: (1) We restrict our model to the space of mobile application user interfaces, and (2) in addition to image and description, we add a third modality, which is the XML representing the layout hierarchy of the mobile application in question, and utilize this to examine how the nature of extracted semantics changes as we examine different representation of a GUI.

Chapter 3

Approach Description

The task we have set out to accomplish is to explore the semantic information encoded into various representations of the GUI of a mobile application. Namely, there are three representations which we take as input: (1) A screenshot of the screen in question, (2) A set of descriptions of that screenshot given in natural language, and (3) the corresponding layout XML dump of the UI hierarchy . At a high level, to accomplish our goal, we must harness the information encoded in these inputs and align their representations appropriately to obtain the best description of screen regions. We obtain these representations by (1) A CNN which utilizes the VGGNet architecture [36], containing about 60 million parameters, (2) A bidirectional recurrent neural network (BRNN) containing over 100 million parameters, and (3) an RNN to map the XML code from the layout dump to reasonable descriptions of the UI.

This chapter will proceed as follows. First, we will outline the procedures around the collection of the dataset. Then, in sections 3.2, 3.3, and 3.4, we will describe in full detail our adaptation of the *neuraltalk2* framework. In section 3.5, we will discuss our procedure for selecting the optimal hyperparameters. Finally, in section 3.6, we will describe our Seq2Seq model which will be trained on the layout hierarchy, and will serve as a frame of reference for the accuracy of Clarity.

3.1 Data-Collection

There exist many datasets for training an image-to-description framework, including but not limited to MSCOCO [23], Flickr8k, and Flickr30k [32]. We argue that these datasets are insufficient for describing regions of mobile application screens because (1) The salient features of images in these datasets do not correspond to features that would be extracted from screenshots of mobile applications and (2) The vocabulary found in the annotations of these datasets is not sufficient to describe mobile application screens. For these reasons, we have constructed a dataset of roughly 14,000 screenshots of mobile applications and 5 corresponding annotations for each. This data collection was crowd sourced using Amazon Mechanical Turk.

Amazon Mechanical Turk workers (Turkers) were assigned Human Intelligence Tasks (HITs) which consisted of 1 image, a screenshot of a running mobile application. Turkers were then asked to give 5 descriptions of the screenshot. The first descriptions was a “high level description”, meaning that the whole screen was to be described, summarizing its functionality, e.g. “A log in screen for Facebook”. The next four annotations were “low level descriptions” in which the Turkers described the functionality of individual screen components or small groups of components. In the datasets mentioned above (MSCOCO and the Flickr datasets), annotations contained references to objects e.g. an elephant, which were likely unique in the image. However, mobile application screens often have multiple components of the same type (there can be many buttons on a screen, each serving very different purposes). With this in mind, we instructed Turkers to give a spatial reference of where on the screen the component in question is, in addition to describing its purpose. An example of a low level description would be “In the top left corner there is a button which returns the user to the previous screen”. Moreover, the Turkers were provided with a handful of examples which were annotated by the author of this paper, in order to give them a proper frame of reference.

The Turkers were paid $0.12 for every image tagged, and data was filtered manually by the author of this paper. Approximately 20% of received submissions were rejected for poor quality. The criteria, in descending order of importance, of how the data was evaluated is listed below.

1. Both the high and low level descriptions faithfully describe the screen

2. For low level descriptions, the functionality attributed to the components matches the judgement of the reviewer

3. For low level descriptions, spatial references were given by the Turker, indicating where on the screen the component or group of components reside.

4. Proper spelling and grammar were used by the Turker

Batches of data were collected at a rate of, on average, 2,000 image-description pairs per week, and an average of approximately four hours was spent on each of these batches of 2,000. Turkers who did not faithfully adhere to the directions, or who gave multiple bad submissions, were immediately banned from the task.

3.2 Image Representations

The goal outlined in this section is to map an input image into the semantic space that is common to all three modalities. Toward this end, we borrow heavily from Karpathy et. al. [19] in the network architecture used and the training method. To summarize, we use the VGGNet architecture [36], loss from the alignment objective (to be discussed below) is backpropagated through the networks for both modalities in the usual way. When complete, images are represented by a set of 20 h-dimensional vectors, where h is the size of the embedding space. A visual for how this set of vectors is constructed can be found in 3.1. More details can be found in [19].

Our work, with respect to representing images, departs from Karpathy et. al in two major ways. First, unlike their work, we train our CNN completely from scratch on

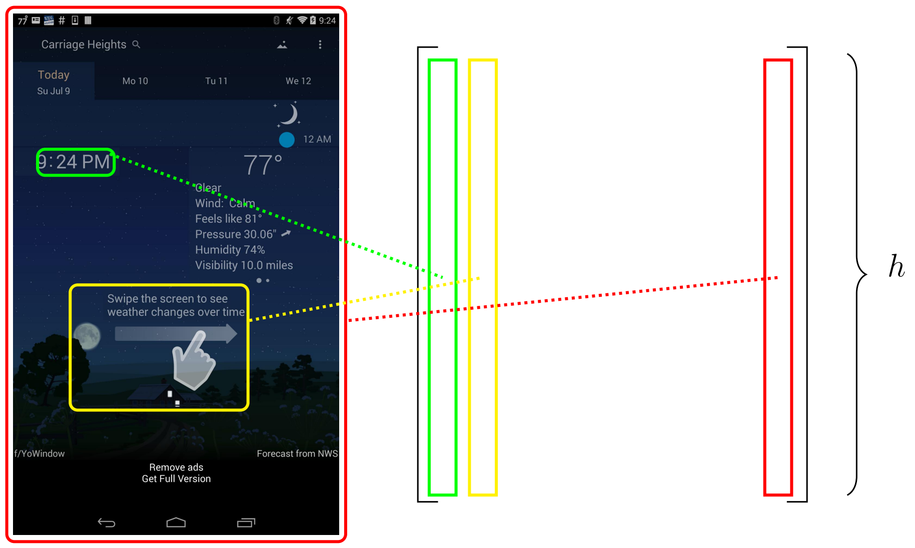


Figure 3.1: How objects are detected within the image and then represented as a set of vectors.

our own dataset. We, again, make the argument that other types of images do not share enough features with screenshots of mobile applications to make transfer learning beneficial. Secondly, we use vastly smaller embedding sizes, which follows from the restricted problem domain. In particular, in the Microsoft COCO dataset, there are 91 object types to be considered [23]. In contrast, Moran et. al [28] identified only 15 categories for leaf-level android components to be sufficiently common for a classifier to consider. Likewise, vocabulary used to describe mobile application GUIs is necessarily smaller than vocabulary used to describe any given object.

In accordance with [19], we follow Girshick et. al [12] and use a Regional Convolutional Neural Network (R-CNN) to extract "region proposals" from the image, and use selective search to choose the 19 top regions which might represent android components, and take the screenshot as a whole as the 20th region. The pixel values corresponding to each of these regions is then the input for a forward-pass through the CNN, by which we obtain a 256-dimensional vector representation of the image region. The screen shot as a whole is then represented by a 20 element set of these 256-dimensional vectors

3.3 Sentence Representation

In the above section, we have mapped the pixels found in the input screenshot to a space which we will henceforth refer to as the "semantic space". We want to map the text corresponding to each training instance to the same semantic space, and then align the two modalities. Ideally, we want to avoid using fixed templates such as those found in [17, 14]. Also, to utilize the most context possible, we want to avoid having fixed context windows, as in [20]. To address these issues, we again follow in the footsteps of Karpathy et. al [19] and use a bidirectional recurrent neural network (BRNN) [34] to embed the text descriptions of the mobile application into the semantic space. The details of this embedding process follow.

3.3.1 Pre-processing

For each text description of each image, we preprocess the data using the Stanford Tokenizer [24], and only include in the vocabulary words which appear above a certain threshold. For our final implementation, we use the 15 word appearances of our threshold . For words that appear less than the threshold, we substitute them in the word sequences with an unknown word flag. We eliminate all punctuation, and convert all letters to lowercase. After preprocessing, we are left with a total of 1473 words in the vocabulary.

3.3.2 The Model

Our bi-directional RNN is similar to [19], since our work is concerned with determining the extendibility of these techniques and models to the domain of software engineering. For convenience, we will reiterate the structure of the model, and comment on where our model departs from Karpathy et. al, though details can also be found in the aforementioned work. Let t with $1 \leq t \leq N$ denote the position of a word in a given sentence. Our goal is to generate an h-dimensional vector representation of the word. The structure of the BRNN is as follows, and note it is identical to that found in [19]:

$$\begin{aligned}
x_t &= W_w \mathbf{I}_t \\
e_t &= f(W_e x_t + b_c) \\
h_t^f &= f(e_t + W_f h_{t-1}^f + b_f) \\
h_t^b &= f(e_t + W_b h_{t+1}^b + b_b) \\
s_t &= f(W_d(h_t^f + h_t^b) + b_d)
\end{aligned}$$

s_t is the final vector representation of the word at position t. $\mathbf{I}_t$ is a column vector with a single 1 at the t position, and zeroes elsewhere. The activation function f is the rectified linear unit (RELU) function. Note the bidirectional nature of the RNN, as we compute two hidden states: h_t^f (the forward direction), and h_t^b (the backward direction). we learn the weights W_e, W_f, W_b, and W_d, and the biases b_e, b_f, b_b, and b_d. We initialize the matrix W_w using a 300-dimensional word2vec [27] weights.

3.3.3 Mapping to the Semantic Space

Let the number of words in a sequence be N, each of these words is encoded using a 1-of-k representation. Using the BRNN, we encode each of these words into an h-dimensional vector, where h is a hyperparameter representing the size of the word embedding. In our experiments, we found that 256 was a reasonable embedding size given the size of our vocabulary. The exact form and corresponding equations of the BRNN can be found in [19].

3.4 Aligning Modalities

In the previous sections, we described how we have mapped image and textual data to the semantic space. Now arises the problem of how we align these two representations in order to produce the best output relative to the ground truth. As in the sections regarding the representations from text and images, we follow in the footsteps of Karpathy et. al [19] in their *neuraltalk2* framework.

Unlike *neuraltalk2*, our supervision is not necessarily or entirely at the level of entire images and their corresponding descriptions. Recall that we instructed our data taggers to give spatial references when describing the functionality of the screenshot of the mobile applications. In accordance with this divergence, we frame the problem of describing images as "pointing out" some of the most important functionality in the screenshot. When framed in this way, we are able to stay consistent with the alignment objective used in *neuraltalk2*. The important ideas of this alignment objective are discussed below, for the details, we refer readers to the aforementioned work.

As already stated, in using the R-CNN, we detect the top 19 locations of objects in an image, and also take the embedding of the whole image. We need to align words in the textual descriptions with their corresponding objects in the screenshot. Following [19], we take the inner product of the word embedding and the detected object embedding to be a measure of similarity. More formally, let the embedding of the i^{th} detected object in an image be denoted v_i, and let the t^{th} word in a sentence be s_t. The measure of similarity between the detected object and the word is given as $v_i^T s_t$. Then to calculate the score for an image-sentence pair we simply add all combinations of objects and sentence fragments:

$$S_{kl} = \sum_{t \in g_l} \sum_{i \in g_k} max(0, v_i^T s_t)$$

where k and l range over all images and sentences in the dataset, respectively.

3.5 Hyperparameter Tuning

For the bimodal screenshot-to-description framework, we performed a random hyperparameter search [4] on 8 different hyperparameter types. Their names and purposes are listed below.

1. **RNN Size** - The size of the hidden layers of the bidirectional RNN
2. **Input Encoding Size** - The size of both the embeddings for the detected objects in the images, as well as the size for the word embeddings.
3. **LM Drop Probability** - The dropout rate for training the Language Model, in order to reduce overfitting
4. **Optimization Technique** - The optimization technique for training the Language Model (e.g. stochastic gradient descent with momentum)
5. **LM learning rate** - The learning rate of the language model
6. **LM decay start** - The iteration at which the learning rate of the language model begins to decay
7. **LM decay every** - This figure represents the interval at which the learning rate of the language model decays by one half
8. **CNN learn rate** - the learning rate for the convolutional neural network which produces the image embeddings

3.2 shows the final values for these hyperparameters. The random search was done on a slightly downsampled training set, with a validation set of 1000 image-description pairs.

3.6 Hierarchy Representation

In order to examine the nature of semantic information embedded into mobile application screens, we compare the previously described model against a seq2seq model which takes as input the XML dump of the user interface hierarchy of the screen. This XML dump is obtained by using the UIAutomator tool. We utilize the seq2seq [5] model with attention to map the tokens found in the XML dump directly to natural language descriptions of the screen, bypassing the intermediate "semantic space".

Hyper-Parameter	Value
RNN Size	256
Input Encoding Size	256
LM Drop Probability	0.6
Optimization Technique	adam
LM Learning Rate	0.0006
LM Decay Start	7000
LM Decay Every	5000
CNN Learn Rate	0.000015

Figure 3.2: Hyper parameter values used for final model

For experimental consistency, We preprocess the data for the target sequences exactly as described in the previous section.

3.6.1 The Model

We use an encoder-decoder model, where the encoder RNN is a bidirectional RNN, to predict the target sequence given a sequence of input tokens [5]. More formally, suppose a set of input tokens $\mathbf{x} = (x_1, x_2, \ldots, x_m)$, then the encoder RNN produces a series of m hidden states $\mathbf{h} = (h_1, h_2, \ldots h_m)$. Since our encoder is bidirectional, we need a way to combine the hidden states of both the backwards and forwards RNNs. We do this via a simple concatenation, i.e. $h_i = [h_i^f; h_i^b]$, where h_i^f is the hidden state for the forwardRNN, and h_i^b is the hidden state for the backward RNN.

The decoder model predicts a sequence of target tokens $\mathbf{y} = (y_1, y_2, \ldots y_k)$ based on the hidden state for the current time step in the decoder RNN d_i and the attention vector c_i. The attention vector can be seen as a weighted average of the hidden states for the encoder RNN, where the states are weighted by importance with respect to the given token. More precisely, the attention mechanism is structured as follows [5]:

$$c_i = \sum_j a_{ij} h_j$$

$$a_{ij} = \frac{f(d_i, h_j)}{\sum_j f(d_i, h_j)}$$

Where a_{ij} is the "attention score", c_i is the context vector at decoder time-step i, and f is some attention function, often an inner product. Finally, the output of the decoder is a distribution over the vocabulary of fixed size V

$$P(y_i | y_1, y_2, \ldots, y_{i-1}, \mathbf{x}) = \text{softmax}(W[d_i; c_i] + b)$$

The model is trained by minimizing the negative log likelihood of the target sequence.

3.6.2 Implementation

To further our exploration of semantic information encoded into representations of the GUI of a mobile application, from the raw XML dump of the screen hierarchy, we parse different sets of data in order to observe the changes in predicted functionality. That is, we do not simply feed the raw XML dump into the encoder-decoder model, and instead refine input based on the desired parameters. There are four trials in total, enumerated below:

1. Text - We parse only the text within the UI component, ignoring all other properties of the widget.

2. Type - We parse only the type of the UI component, e.g. *LinearLayout*, *Button*, or *TextView*

3. Text and Type - We combine the information of the component type, and the text found in the component

4. Text, type, and location - In addition to the type of the UI component and the text found therein, we also parse the location of where it resides on the screen. That is, we give the bounding box of the component given in the form (x_1, x_2, y_1, y_2)

3.6.3 Motivation Behind Seq2Seq

One research question that our work seeks to answer is to describe the semantic information embedded into a graphical UI. In other words, we want to see to what extent it is possible to know the intended functionality of a user interface by just looking at the screenshot, not having any access to the source code. The XML dumps obtained through UIAutomator serve to form a frame of reference. What we want is to examine the changes in semantic information extracted when we consider text, component types, component bounding boxes, etc, and how these compare with functionality discovered by Clarity.

Chapter 4

Experimental Design

As stated above, the goal of the experiments conducted is to ascertain and somehow quantify the semantic information embedded into the screenshot of a mobile application. To do this, we must address two major concerns:

1. We must find metrics which are suitable for describing the accuracy of the generated functional descriptions of the screenshot, so that we may compare the output of Clarity with the Seq2Seq approach which we have selected as the baseline.

2. Since descriptions of functionality are given in natural language, we must determine the usefulness of the descriptions to humans who have expertise in mobile development.

To address these questions, we turn to the benchmark *BLEU score* [31], which seeks to simultaneously judge descriptions based on the accuracy of its content and by the fluency, or human-usability, of the description. We begin by giving a concise synopsis of the BLEU score, its meaning, and the details of its calculation. Then, in research question 1, we outline how we plan to demonstrate that semantics can be learned from mobile application GUIs similarly to how they are learned in image-captioning frameworks for real-world items. Finally, in research question 2, we give a plan for determining how changing representations for a GUI of a mobile application can effect the quality and nature of the semantics that can be learned.

4.1 Quantitative Evaluation

There are a miriad of metrics that are used to judge the quality of a generated iamge caption, including but not limited to BLEU [31], CIDEr [41], METEOR [1], and ROUGE [22]. For our work, we utilize the most common metric: BLEU. What follows is a description of BLEU and the details of its calculation.

4.1.1 BLEU Score

The BLEU score is a metric which has a very straighforward motivation: the closer a machine translation is to a professional human translation, the better it is [31]. Originally developed to be a metric to measure the accuracy of machine translation with respect to natural languages, BLEU has emerged as a benchmark metric for image-to-description frameworks. This work's closest relative, Karpathy et. al, [19], uses it as the principle metric for measuring the accuracy of image descriptions. We follow suit, for the sake of consistency.

4.1.1.1 The N-Gram Computation

An N-gram is a sequence of N words. To compute the the BLEU-N score for a candidate sentence and a reference sentence, we simply count the total matches between the two sentences of sequences of length N, and divide by the total number of N-length sequences in the reference sentence. More formally, let c be the candidate sentence, and let r be the reference sentence. Let A be the total number of matches, and B be the total number of sequences of N length in the reference sentence, then:

$$\text{BLEU} - N(c, r) = \frac{A}{B}$$

4.1.1.2 Implementation

We compute scores for $1 \leq N \leq 4$, and take the average of the scores as the final BLEU score. Since we have multiple candidate sentences (between 16 and 19), we take the average BLEU score of all candidate sentences. Since we have multiple reference sentences, we match a candidate sentence with its best-match reference sentence.

4.2 RQ$_1$: Comparison to real-world semantics

As discussed before, captioning of images containing real-world objects is a well-studied problem, and promising strides have been made recently [38, 30, 17, 19]. That is, these approaches seek to learn the semantics of objects in the real world, with descriptions like "man riding elephant". Our approach, instead, seeks to learn the semantics of static GUI components, but the goal of determining the underlying semantics of what appears in the image remains the same.

One way to form a frame of reference for the nature of semantics encoded into the GUIs of mobile applications is to compare them against the already established work concerning real-world objects. We compare our BLEU scores to that of recently published work concerning image captioning of real-world objects. In particular, we compare our work to the results of karpathy et. al [19] and Kiros et. al [20]. In RQ$_1$, we seek to answer the question, *How well do the semantics encoded into the GUIs of mobile applications capture functional features that can be translated to natural language descriptions?*

4.3 RQ$_2$: Comparison Between Screen Representations

One of the core goals of this work is to compare and contrast the semantic information encoded into varying representations of the GUI of a mobile application. Particularly, we examine the difference between the image-description framework we call Clarity, and the 4 representations of the XML dump of the layout hierarchy discussed in section 3.6. To remain consistent with the evaluation discussed in section 4.1, we use the BLEU score as our vehicle of comparison. By comparing results from these different representations, we hope to find insight into what sorts of information is lost or gained when viewing a screenshot of an application, versus looking at the source code of the layout hierarchy. In summary, the goal of RQ_2 is to answer the question, *What is the nature of the semantic information encoded into representations of the GUI of a mobile application, and do different representations of a mobile application GUI capture different aspaects of functional information about the underlying software?*

Chapter 5

Results

In this section we seek to answer the questions outlined in section 4.2 and section 4.3. We do this by primarily comparing (1) BLEU scores with state-of-the-art image captioning work and (2) by comparing BLEU scores obtained from the different GUI representations against one another to determine their quality.

5.1 $\mathbf{RQ}_1$

Before discussing comparisons between state-of-the-art work on image captioning and our approach, we must address certain limitations to these comparisons. It often requires many words to describe the full functionality of a mobile application screen, whereas the core semantics of an image depicting real-world objects can be described in relatively fewer. In addition, there are often many more important "low level" aspects of a mobile application screen, which must be meticulously described to form a clear picture. For these reasons, learning the semantics of a mobile application screen can be more challenging. On the other hand, visual features of mobile application screens are often more discernable than objects in the real world, owing to the restricted number of discrete categories whose features must be learned.

Calculations for the BLEU scores were carried out on a test set of 1,000 images. We record the best caption (meaning that the produced caption can match either the "high

	BLEU-1	BLEU-2	BLEU-3	BLEU-4
Clarity	0.566	0.367	0.251	0.166
Karpathy et. al	0.579	0.383	0.245	0.160
Kiros et. al - MLBL-B-Conv	0.349	0.165	0.085	-
Kiros et. al - LBL	0.327	0.144	0.068	-

Figure 5.1: Comparison of Clarity with image-captioning frameworks for real-world objects. MLBL-B-Conv refers to a Modality-Based Log Bilinear Model with a convolutional network, and LBL is a baseline Log Bilinear Model

level" description or any of the four "low level" descriptions. The results are given in figure (5.1).

In figure 5.1, we compare Clarity with 3 other image-captioning frameworks which concern themselves with real world objects. The first is the closest relative of our work, Karpathy et al [19]. The third and fourth are the work of Kiros et al. [20]. As is observed in the table, we achieve comparable results with Karpathy et. al, even achieving slightly better results in longer word-grams, speaking to the *fluency* of the descriptions [31]. However, when comparing these approaches, there are very important caveats which must be kept in mind:

1. **Data Set Content** - Our dataset is structured fundamentally differently than the image-captioning frameworks seeking to describe real world objects. While in their datasets, each description should more or less agree agree with one another, in ours, each description describes a very different aspect of the screen. Therefore, in our case, there is no guarantee that a high BLEU score indicates a *complete* description of the screen.

2. **Image Categories** - As previously discussed, real world datasets have a much more diverse set of image categories than datasets oriented toward mobile application GUIs, such as that found in Moran et. al [28]

3. **Descriptions** - The dataset used by Kiros et. al only had between 1 and 3 descriptions, compared to 5 for both our dataset and MSCOCO [23]

Our goal for comparing against image-captioning frameworks concerned with real-world

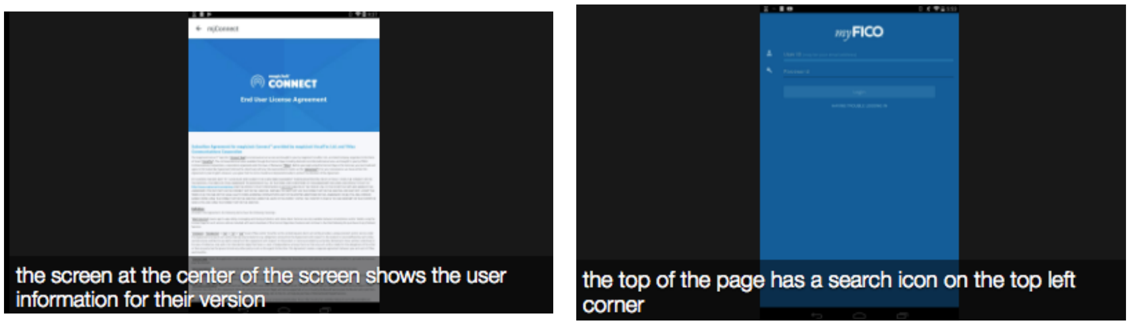

CONNECT
End User License Agreement
the screen at the center of the screen shows the user information for their version
myFICO
the top of the page has a search icon on the top left corner

Figure 5.2: Example output of clarity. Note that spatial references are given. We chose these examples to give a representative snapshot of the capabilities and shortcomings of clarity.

objects is *not* to attempt an apples-to-apples comparison, or to assert the superiority of our model over others. We hope instead, to simply demonstrate that semantic information can be learned reasonably well from an image representation of mobile application GUIs. After the preceeding analysis, we can answer RQ$_1$ in a positive way: **Clarity captures the semantics of screenshots of mobile application GUIs well, with respect to similar image-captioning tasks**. Examples of output can be found in 5.2.

5.2 RQ$_2$

Comparing Clarity to the Seq2Seq model also comes with certain limitations. First and foremost, the Seq2Seq model has largely no way of enumerating different "low level" components, whereas the CNN-BRNN model uses first detects objects in the image, and uses only the pixels found therein to detect features of the image. This leads to desriptions that are specific to the object found. While the attention mechanism in the Seq2Seq model should alleviate a portion of this concern, especially when the sequence representation of the XML dump contains bounding boxes, our experiments reveal that it is not enough. Subsequently, in order to make a decent comparison, we compare each model taking only "high level descriptions" as inputs. Clarity performs well in this experiment, whose results can be found in figure 5.3.

	BLEU-1	BLEU-2	BLEU-3	BLEU-4
Clarity	0.541	0.406	0.299	0.208
Seq2Seq - Text Only	0.305	0.081	0.038	0.026
Seq2Seq - Type and Text	-	-	-	-
Seq2Seq - Type, Text, and Bounding Box	-	-	-	-

Figure 5.3: Comparison of ReDraw with the Seq2Seq model which uses as input the parsed UI XML dumps.

Chapter 6

Conclusion

In this work we have presented a deep-learning approach to describing the functionality of a mobile application's GUI, and explored the nature of the semantics of GUI representations in general. We have done this by (1) creating a new dataset, specifically designed for the task of describing functionality of screen components, (2) constructed a CNN-BRNN model which identifies the functionality of a screen and its components by considering a GUIs graphical representation, and (3) trained a Seq2Seq model which maps the XML dump of the GUI's layout hierarchy to descriptions of screen functionality. We have demonstrated that useful semantics can be extracted from these representations, especially with respect to similar approaches which seek to describe semantics of real-world images.

6.1 Limitations and Avenues for Improvement

While this work aims to explore the relationship between screen semantics (functionality) and representations of the GUIs of mobile applications, there is much room for improvement. Although the CNN-BRNN model achieves very good results for the image-captioning problem, there are a host of other approaches which may work well for our specific problem domain: describing the functionality of application components. It may, perhaps, be the case that template-oriented designs work well for this task. Indeed, while we contend that taking a deep learning approach is a reasonable one given its success in closely related problems, often simpler, more lightweight approaches can achieve similar or even superior results [10]. We leave a more exhaustive comparison between deep learning and other techniques to future work.

Furthermore, the Clarity dataset is, at the time of the writing of this paper, the only dataset available for taking a learning approach to describing GUI semantics. Alternative structures and forms of a dataset may be more conducive to providing useful descriptions. In particular, the Clarity dataset aims to be as *wide* as possible. That is, we attempted to tag as many different images as possible so that the model can generalize to a wide range of GUIs. It is worth considering, instead, to iterate over the same screen (or screen component) multiple times to form a consensus between tags, i.e. a more *tall* approach.

6.2 Future Work

Evaluations in this work were metrics-driven, utilizing the benchmark BLEU score. However, the BLEU metric has been repeatedly called into question regarding its relationship to descriptions that are useful to humans [6]. To remedy this, to take not only a *quantitative* approach, but also a *qualitative* approach. This may take the form of, for instance, a survey to experts in mobile development, who would address questions of usefulness and completeness of the generated descriptions of functionality.

We also plan on future work to address the model itself. For Clarity, we use randomly initialized weights in both the CNN and BRNN, and train them in tandem, from scratch. In contrast, Karpathy et al [19] pre-train their CNN on ImageNet [7], a large dataset of labelled images for classification. Pretraining Clarity's CNN on the analagous dataset for our domain, such as that presented in Moran et. al [28], could be beneficial to performance.

Bibliography

[1] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005.

[2] Kobus Barnard, Pinar Duygulu, David Forsyth, Nando de Freitas, David M Blei, and Michael I Jordan. Matching words and pictures. *Journal of machine learning research*, 3(Feb):1107–1135, 2003.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[4] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[5] Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*, 2017.

[6] Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Re-evaluation the role of bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[8] Sabeen Durrani and Qaiser S Durrani. Applying cognitive psychology to user interfaces. In *Proceedings of the First International Conference on Intelligent Human Computer Interaction*, pages 156–168. Springer, 2009.

[9] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *Advances in neural information processing systems*, pages 2121–2129, 2013.

[10] Wei Fu and Tim Menzies. Easy over hard: a case study on deep learning. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 49–60. ACM, 2017.

[11] Margaret M Gardiner and Bruce Christie. *Applying cognitive psychology to user-interface design*. John Wiley & Sons, Inc., 1987.

[12] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[13] Jun Gong and Peter Tarasewich. Guidelines for handheld mobile device interface design. In *Proceedings of DSI 2004 Annual Meeting*, pages 3751–3756, 2004.

[14] Ankush Gupta and Prashanth Mannem. From image annotation to image description. In *International Conference on Neural Information Processing*, pages 196–204. Springer, 2012.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR'16, pages 770–778, June 2016. ISSN:.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 47:853–899, 2013.

[18] Andrej Karpathy. *Connecting Images and Natural Language*. PhD thesis, Stanford University, 2016.

[19] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[20] Ryan Kiros, Ruslan Salakhutdinov, and Rich Zemel. Multimodal neural language models. In *International Conference on Machine Learning*, pages 595–603, 2014.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, pages 1097–1105. Curran Associates, Inc., 2012.

[22] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft

coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[24] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[26] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

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

[28] K. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk. Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps. *ArXiv e-prints*, February 2018.

[29] Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori Pollock, and K Vijay-Shanker. Automatic generation of natural language summaries for java classes. In *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*, pages 23–32. IEEE, 2013.

[30] Vicente Ordonez, Girish Kulkarni, and Tamara L Berg. Im2text: Describing images using 1 million captioned photographs. In *Advances in neural information processing systems*, pages 1143–1151, 2011.

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

[32] Cyrus Rashtchian, Peter Young, Micah Hodosh, and Julia Hockenmaier. Collecting image annotations using amazon's mechanical turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pages 139–147. Association for Computational Linguistics, 2010.

[33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vision*, 115(3):211–252, December 2015.

[34] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[35] Ben Shneiderman, Catherine Plaisant, Maxine Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. *Designing the user interface: strategies for effective human-computer interaction*. Pearson, 2016.

[36] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1556, 2014.

[37] Richard Socher and Li Fei-Fei. Connecting modalities: Semi-supervised segmentation and annotation of images using unaligned text corpora. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 966–973. IEEE, 2010.

[38] Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association of Computational Linguistics*, 2(1):207–218, 2014.

[39] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[40] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, CVPR'15, 2015.

[41] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575, 2015.

[42] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, pages 818–833. Springer International Publishing, Cham, 2014.