Visual Sketching: From Image Sketches to Code

Marcelo d'Amorim
Federal University of Pernambuco
damorim@cin.ufpe.br

Rui Abreu
INESC-ID and IST, U.Lisbon
rui@computer.org

Carlos Mello
Federal University of Pernambuco
cabm@cin.ufpe.br

Abstract

Writing code is difficult and time consuming. This vision paper proposes Visual Sketching, a synthesis technique that produces code implementing the likely intent associated with an image. We describe potential applications of Visual Sketching, how to realize it, and implications of the technology.

Keywords

code synthesis, machine learning, computer vision

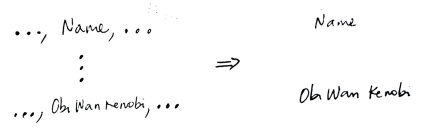
ACM Reference Format:
Marcelo d'Amorim, Rui Abreu, and Carlos Mello. 2020. Visual Sketching: From Image Sketches to Code. In *The 42th International Conference on Software Engineering, May 23–29, 2020, Seoul, South Korea.* . ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Writing code is challenging. Programmers make mistakes for different reasons. A variety of languages, libraries, and coding tools are available today to solve different problems. However, keeping up with all the choices available can be daunting. Furthermore, development is often a repetitive task, leading to copies of similar code and errors [16].

We propose *Visual Sketching*, a low/no-code approach [3] to mitigate some of these problems. Visual Sketching uses computer vision algorithms and machine learning to translate images into code. With Visual Sketching developers draft their intent with a picture and obtain rapid access to code implementing their specification. The idea of Visual Sketching applies to any domain, but we argue that the approach is particularly useful for data scientists. For example, it has been reported that data scientists spend ~80% of their time preparing the data that will be analyzed [12]. Visual Sketching can help reducing this cost.

Visual Sketching is a technique for program synthesis, an area that is mainly investigated by the Programming Languages and Software Engineering communities, but one that is intensively fertilized by Machine Learning. Although this is a very active area of research [13], to the best of our knowledge, the use of images to compile code snippets has not been explored in depth. Model-based development uses visual languages (e.g., Simulink and Stateflow [7]) to generate code. Microsoft's PROSE [6] allows users to define their own domain-specific language to synthesize code. In contrast to these approaches, Visual Sketching synthesizes code from images; it has been inspired by coding cards [5], which has shown to be a useful tool to teach programming languages [2, 9]. It innovates on that idea by allowing visual patterns to describe code intent.




```
import pandas as pd
fileName = ? #fill hole (in)
df1 = pd.read_csv(fileName)
colName = ? #fill hole (in)
df2 = df1[[colName]]
file = ? #fill hole (out)
df2.to_csv(file, index=None)
```

Figure 1: Column projection from csv file.

The Workflow: The Visual Sketching workflow consists of the following steps. First, a developer defines a transformation from images into code. The transformation is a function that takes an image on input and looks for a matching image for the input image on an image database. If a match is found, the transformation extracts relevant information from the input image, if needed, and generates code on output that is associated with the reference image. Developers that define these transformations are familiar with computer vision algorithms and the infrastructure we provide; they are not (necessarily) users of the tool. Second, users write an image describing their intent. Third, the tool looks for a match and if a match is found it produces the code. Fourth, the user fills the "holes" in the code. A hole denotes elements that were not inferred from the input image and should be completed by the user.

2 Motivating Examples

This section motivates Visual Sketching with two examples of tasks commonly performed by data scientists.

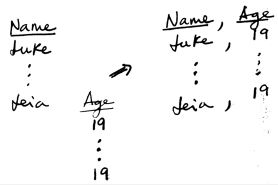
2.1 Data Wrangling

Data Wrangling is an important initial step in data analysis that consists of preparing the data. It is considered a very time-consuming and tedious step [12]. The following examples show how Visual Sketching can help data wrangling.

Consider the functionality of projecting a column from a comma-separated values (csv) file. Figure 1 illustrates the reference image and the corresponding code associated with this transformation. The user-provided image is the input to the transformation; it does not appear in the figure. The image on the figure is a reference to match against the input image. The input image should have similar layout structure compared to the image in the figure (see

ICSE '20, May 23–29, 2020, Seoul, South Korea

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn




```
import pandas as pd
file1 = ? #fill hole (in)
df1 = pd.read_csv(file1)
file2 = ? #fill hole (in)
df2 = pd.read_csv(file2)
df3 = pd.concat([df1,df2],axis=1,sort=False)
filename = ? #fill hole (out)
df3.to_csv(filename, index=None)
```

Figure 2: Merging columns from two files with aligned data.

Section 3.1). Considering the code, note that it contains holes, expressed with "?", that should be filled by the developer. The code in the figure uses the Python Pandas library, which is very convenient for the task of manipulating matrices. For this example, the developer needs to (1) indicate the input file name and to (2) indicate the column name he wants to project. Sections 3.2 and 3.3 shows how to identify the name of the column from the input image, which could be used to bind the actual column name the user desires to project and consequently avoid the need to create the hole for `colName`.

Figure 2 shows another wrangling operation. This time the goal is to merge two different files into one file. The data in those files is aligned. This example shows that we are not assuming that users would write descriptions the same ways. Various kinds of differences can emerge. These are aspects that need to be taken into account by the vision algorithms (Section 3).

2.2 Design of Neural Nets

Neural nets are an increasingly popular technology that enables machines to learn. As in the previous example, data scientists spend reasonable amount of time designing different network architectures for different problems[1]; Visual Sketching can help reduce this burden. The example on the left-hand side of Figure 3 shows two alternative visual sketches of a very simple neural net architecture[2]. This example highlights the power of Visual Sketching to produce complex code based on visual descriptions. Many other network architectures are possible. It is also worth noting that the Asimov Institute [1] categorized various of these architectures and standardized symbols to denote different kinds of neurons/nodes [4]. We plan to take that into consideration to support rapid prototyping of those nets.

3 Document Image Analysis - Segmentation and Recognition

Visual Sketching needs to process images. As the image sketches are essentially a composition of text elements, they can be treated as documents [10]. Document Engineering is the research field that studies how to process documents; it includes several specialized algorithms for analyzing this kind of file (image or not). More in detail, Visual Sketching executes three steps to analyze an image: (1) structural analysis of the document, (2) document image segmentation, and (3) text recognition.

3.1 Structural Analysis

Let us consider the example from Figure 1 for illustration. Structural analysis is an important step to define the type of structure of the document for further processing. In this case, an image similarity index could be considered. In particular, note that there are several off-the-shelf solutions for that task, but in this case many aspects can vary in the document image, e.g., the writing style, position of the text, colors. Therefore, similarity algorithms based on the structure of the document are more suitable for this purpose.

The first step in the structural analysis is thresholding the document image. Thresholding is a common operation in document image analysis where the input image—usually in greyscale—is converted into a black-and-white image. A threshold value defines which colors are converted into white (the background) or into black (the foreground), corresponding to the text. For this purpose, we plan to use the algorithm proposed by Howe [14], winner of the Document Image Binarization Contest (DIBCO) in 2013 [18]. As this operation can lose part of the characters, a flood fill algorithm, as presented in [22], is needed. This algorithm closes small gaps within black areas which is useful if part of the character is lost because of thresholding. With the image represented in black-and-white, the method proceeds to analyze the structure of the image. For example, considering Figure 1, the method recognizes the original data on the left, an arrow delimiter in the middle of the figure, and the final text at the right column. Thus, in this case, we are looking for a three-column structure.

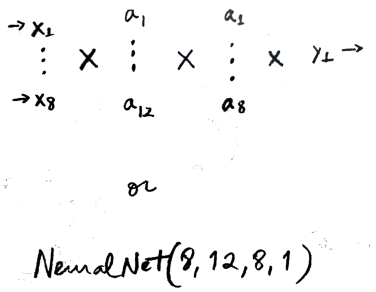
A morphological operation of closing [11] is applied to group sequence of characters into a single object. As we are searching for the basic structure of the document, details of the text can be ignored (e.g., points and commas). After a connected components analysis [19], the components with less than 500 pixels are removed [3]. This also removes noise elements.

A vertical projection profile is then performed to separate regions with text from the regions without text. The profile counts the number of black pixels in each column of an image. Columns with a small number of black pixels indicate a column with no text and it is labeled as such. Groups of few adjacent no-text columns between regions of text are converted into text regions as they probably contain parts of a character. On the other hand, groups of few adjacent areas labeled as text are converted into no text areas as they probably contain noise or non-relevant characters (as points or commas). After this projection profile step, regions have been identified and classified as either textual or non-textual areas.

As mentioned before, only three regions would be produced for the visual sketch from Figure 1. At this point, these regions define the structure of our sketch and could be used for image similarity. Recall that, although off-the-shelf image similarity libraries could be used, they would certainly perform poorly given the variety of characteristics in human writing. Also, it must be said that other

[1] A senior researcher from Google UK reported to us in a personal communication that up to 40% of his time is spent on this task.
[2] Based on an example that uses eight different parameters for diabetes classification [8]

[3] This value was also defined empirically and is related to the stroke width of the text and can be defined automatically [17].




```
from keras.models import Sequential
from keras.layers import Dense
# define the model
model = Sequential()
act = ? #option: 'relu', 'sigmoid', ...
model.add(Dense(12, input_dim=8, activation=act))
act = ? #option: 'relu', 'sigmoid', ...
model.add(Dense(8, activation=act))
act = ? #option: 'relu', 'sigmoid', ...
model.add(Dense(1, activation=act))
# compile the keras model
myloss= ? #option: 'binary_crossentropy', ...
myopt= ? #option: 'adam', ...
model.compile(loss=mils, optimizer=myopt, metrics=['accuracy'])
# load the training data
X = ? #inputs, NumPy (list of arrays)
y = ? #output, Numpy array
# fit & evaluate
model.fit(X, y, epochs=150, batch_size=10)
_, accuracy = model.evaluate(X, y)
```

Figure 3: Neural net with 8 inputs, one output, and two hidden layers—one with 12 neurons/nodes and the other with 8 nodes. All layers are fully connected, as the cross suggests.

structures will require other processing; this will be addressed in the future.

3.2 Document Image Segmentation

Let us consider a case where Visual Sketching needs to extract some part of the text from the input image. For the sake of illustration, let us consider the string "Name" that appears at the top right of Figure 1 needs to be extracted. Image segmentation is concerned with identifying parts of the image that contain elements of interest.

Considering our example from Figure 1, the image has already been split in three regions, the system works on the rightmost region where that element is to be located. For this purpose, the area is separated from others and the image is complemented. This is a necessary step for the application of mathematical morphology algorithms. An operation of dilation with a 20-pixels disk as structural element is then applied, creating a large region where the original text is contained. In a simple way, dilation is a morphological operation that increases the area of a group of white pixels in an image according to the structural element. For example, a white single pixel could become a 3x3 group of white pixels through dilation with a 3x3 square as strucutral element. We can resemble the idea of a real dilation of an object (in opposition to an erosion operation that decreases the area of an object). A bounding box is defined as the minimum rectangle that contains an object. Bounding boxes are created for every object in the region. For our purpose, we select the uppermost box.

Figure 4 summarizes the process so far step-by-step on a similar example to the one of Figure 1. Figure 5 shows the extraction of text for different examples. Note that the extraction of the document structure is a necessary step for extracting the text and such structure is key to implement image similarity. As it can be seen in this example, if the letters are close enough, they will be considered as part of the same word.

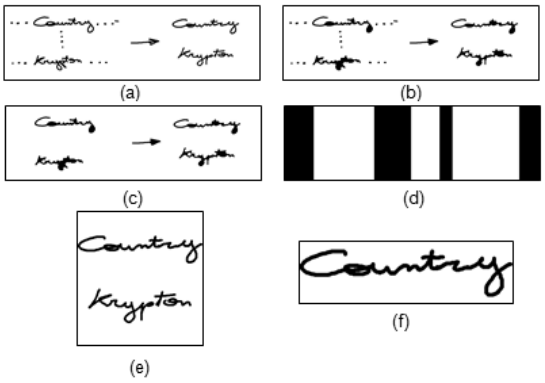


Figure 4: Segmentation major steps: (a) Original sketch (already converted into a black and white image; (b) resulting image flood fill and closing operation; (c) result after removal of small elements; (d) final projection profile image (white areas corresponds to textual areas); (e) last column of textual area; and (f) final segmented and selected text area.

3.3 Text Recognition

The last step is to recognize text segments, if necessary. This part of the system is still under development in our prototype implementation, but we can leverage recent advances in deep learning to precisely recognize text from drawings. Some promising results can be found in [21][23][15].

4 Discussion

We make the following observations about Visual Sketching:

- Visual Sketching is not a silver bullet to avoid coding. It is more useful in domains of application where it is more natural to express intentions visually. The technique makes the hypothesis that developers can complete the code associated with the visual sketches. Conceptually, that is the inverse of the logic used in Program Sketching [20] where developers are asked to write the sketch of the code and a code synthesizer is used to complete the holes left in the incomplete program.

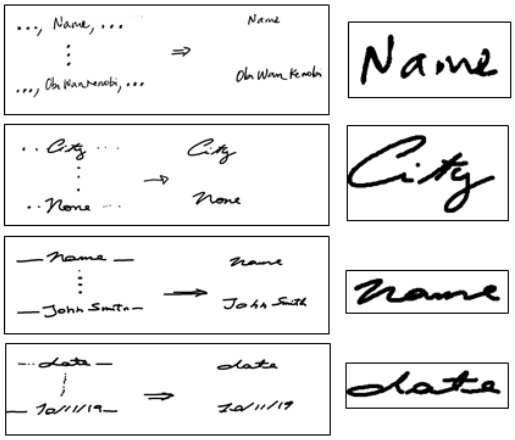


Figure 5: (left column) Original sketches; (right column) final text segmentation for further recognition.

- The wide adoption of development-oriented discussion forums, such as StackOverflow, contributes to our approach rather than competes with it. The identification of common patterns from these forums can be used to feed the sketch database that we plan to build.
- All the examples presented in this proposal produce code from templates, rather than specifications [13], once a pattern is detected. An additional functionality of Visual Sketching is to identify more general patterns, such as arbitrary neural networks with custom symbols as those indicated by the Asimov Institute [1].
- We chose Python as the target programming language to demonstrate Visual Sketching given its popularity in the data sciences, web development, and various other domains. However, the decision of programming language is not a central aspect of the research. The biggest challenge is to identify developer's intention from visual sketches. In principle, other programming languages can be supported by adding new template mappings.
- An assumption that we make with practical implications on Visual Sketching is that the developer can easily transfer images to his development environment. One alternative for that is to use a digital pen or take a picture from a sheet of paper.

The main challenges of this research effort include:

- **Tension between simplicity and expressiveness.** Deciding the right balance between simplicity and expressive power of the visual sketches is one important challenge. We should determine with care how much details of the code should be abstracted from the developer. If too general, Visual Sketching becomes a Turing-complete language not distinct from modern program languages. If too specific, the developer may prefer to look for a solution on the web as she would need to familiarize herself with the language. We sincerely believe that deploying a fully-functional prototype on the web can help decide this balance with the input from the crowd.
- **Selection of Algorithms and Efficiency.** There is a vast array of options of computer vision algorithms to use for solving each of the problems we presented. Our plan is to first support specific tasks as those described above and only later expand to let users of our platform describe their own transformation patterns.

5 Conclusions

This paper presents Visual Sketching, a bold vision for a new research direction on code synthesis. Visual Sketching is not yet supported by solid results, but rather by preliminary results and a strong intuition by the authors. We argue that we have discussed an ambitious idea that requires synergies from a multidisciplinary team in order to be successful. In particular, it integrates the Programming Languages, Software Engineering, and Computer Vision communities. Successfully achieving this bold vision would lead to an impact on education and development productivity, mainly.

Acknowledgments. This research was partially funded by INES 2.0, FACEPE grants PRONEX APQ 0388-1.03/14 and APQ-0399-1.03/17, CAPES grant 88887.136410/2017-00, and CNPq grant 465614/2014-0. At Portugal, the research was funded through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 and PTDC/CCI-COM/29300/2017.

References

[1] [n.d.]. The Asimov Institute website. https://www.asimovinstitute.org.
[2] [n.d.]. Brainscape. https://www.brainscape.com/subjects/computer-programming-flashcards.
[3] [n.d.]. The Low-Code/No-Code Movement: More Disruptive Than You Realize. https://tinyurl.com/yynmkxu5.
[4] [n.d.]. The Neural Net Zoo. https://www.asimovinstitute.org/neural-network-zoo/.
[5] [n.d.]. Play Cards. Learns How to Code. http://codecards.io/.
[6] [n.d.]. PROSE SDK. https://microsoft.github.io/prose/.
[7] [n.d.]. Stateflow website. https://www.mathworks.com/products/stateflow.html.
[8] [n.d.]. Your First Deep Learning Project in Python with Keras. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/.
[9] A. Baker, E. O. Navarro, and A. van der Hoek. 2003. Problems and Programmers: an educational software engineering card game. In *ICSE*. 614–619.
[10] Carlos Alexandre Barros De Mello, Adriano Lorena Inacio de Oliveira, and Wellington Pinheiro Dos Santos. 2012. *Digital document analysis and processing*. Nova Science Publishers.
[11] E.R. Dougherty. 1992. *An introduction to morphological image processing*. SPIE Optical Engineering Press.
[12] Tim Furche, Georg Gottlob, Leonid Libkin, Giorgio Orsi, and Norman W. Paton. 2016. Data Wrangling for Big Data: Challenges and Opportunities. In *Proceedings of the 19th International Conference on Extending Database Technology (EDBT)*. 473–478.
[13] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. 2017. Program Synthesis. *Foundations and Trends in Programming Languages* 4, 1-2 (2017), 1–119. https://doi.org/10.1561/2500000010
[14] Nicholas R Howe. 2013. Document binarization with automatic parameter tuning. *International Journal on Document Analysis and Recognition (IJDAR)* 16, 3 (2013).
[15] R Reeve Ingle, Yasuhisa Fujii, Thomas Deselaers, Jonathan Baccash, and Ashok C Popat. 2019. A Scalable Handwritten Text Recognition System. *arXiv preprint arXiv:1904.09150* (2019).
[16] Kapser, Cory. 2009. Toward an Understanding of Software Code Cloning as a Development Practice. http://hdl.handle.net/10012/4753
[17] Shijian Lu, Bolan Su, and Chew Lim Tan. 2010. Document image binarization using background estimation and stroke edges. *International Journal on Document Analysis and Recognition (IJDAR)* 13, 4 (2010), 303–314.
[18] I. Patrikakis, B. Gatos, and K. Ntirogiannis. [n.d.]. ICDAR 2013 Document Image Binarization Contest (DIBCO 2013). 2013 12th International Conference on Document Analysis and Recognition (2013).
[19] Robert Sedgewick. 1900. *Algorithms in C++, 3/e*. Pearson Education India.
[20] Armando Solar-Lezama. 2008. *Program Synthesis by Sketching*. Ph.D. Dissertation. Berkeley, CA, USA. Advisor(s) Bodik, Rastislav. AAI3353225.
[21] C. Tensmeyer and C. Wigington. [n.d.]. Training Full-Page Handwritten Text Recognition Models without Annotated Line Breaks. ICDAR 2019. https://arxiv.org/abs/1909.02576.
[22] Shane Torbert. 2016. *Applied computer science*. Springer.
[23] S. Xiao, L. Peng, R. Yan, and Wang. S. [n.d.]. Deep Network with Pixel-Level Rectification and Robust Training for Handwriting Recognition. ICDAR 2019. https://arxiv.org/abs/1909.02576.