AI Assisted User Interface Development

CS297 Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Class

CS 297

By

Siddharth Kulkarni

December, 2018

TABLE OF CONTENTS

Abstract

Converting a visual user interface design created by a designer into computer code is a typical job of a user interface engineer in order to develop beautiful web and mobile applications. This conversion process can often be extremely tedious, slow and prone to human error. In the coming years technology such as deep learning will enable the design of expressive and intuitive products while eliminating hurdles from the product development process. This report aims to understand the various artificial intelligence assistant techniques explored by researchers in this field. It will help streamline and automate the overall product development routine by generating platform ready prototypes directly from design sketches.

Index **terms – Artificial Intelligence (AI), machine learning (ML), user interface (UI), Convolutional Neural Networks (CNN)**

I. Introduction

The User interface design process involves translating project requirements provided by customers and project managers to creative explorations in the form of mockups. These mockups are tested and finally converted to working prototypes by a User Interface Engineer. A lot a creativity that starts on a whiteboard where designers share ideas is translated into working HTML wireframes by a developer to play within a browser. Each of these steps takes significant domain specific expertise and effort which delays the design process. My deep learning powered solution aims to streamline overall user interface development by generating prototypes directly from design sketches.

Only recently has the field of automatic program synthesis from visual, audio, textual inputs using machine learning techniques become popular. Though the generation of computer programs is an active research field, Program Synthesis from visual inputs is still a nearly unexplored research area. The closest related work is a method developed by Nguyen et al, to reverse-engineer Android user interfaces from screenshots. A number of research papers and literature have shown that deep neural networks are able to learn to describe objects in an image and their relationships with textual descriptions [1]. Most of these methods primarily use a *Convolutional Neural Network (CNN)* to map raw images into a learned representation using unsupervised learning. This is usually followed by the use of a *Recurrent Neural Network (RNN)* which perform language modeling on the textual description associated with the input picture using gradient descent for optimization [1].

Airbnb Researcher from the Google Brain team were able to train a machine to produce captions to describe images. The algorithm actually tied for first place at the Microsoft COCO 2015 image caption challenge. They even went on to release the source code as a module for Tensorflow for public use and development. Their system used the inception v2 image classification model with an accuracy of 98%. Recently an inception v3 has also been released with significant performance and under the hood improvements.

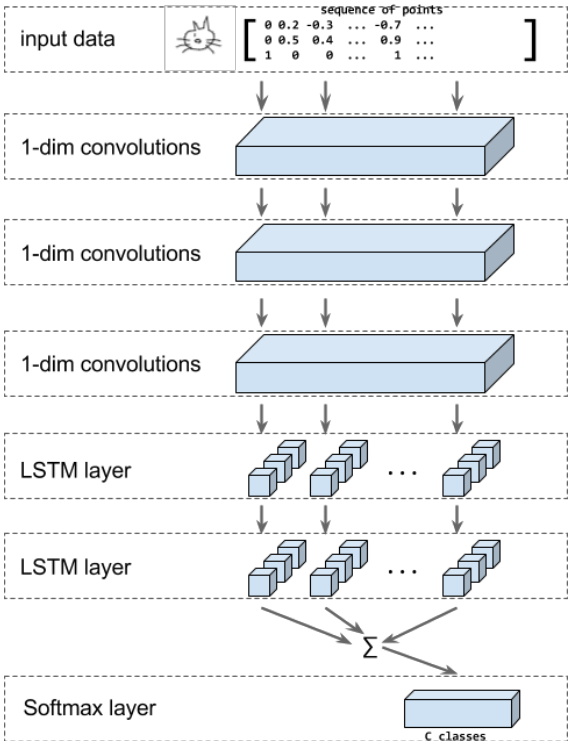
Pix2code [1] aims to generate from pixel values, variable length strings of text or tokens to form image captions. Tony Beltramelli [1] divides the problem into three subtasks: a computer visual problem, a language model and finally combining the results from previous two steps to infer scene understanding and object position and pose modelling. The author uses CNN's to solve the computer vision problem. CNN's perform well for unsupervised learning and can map feature set learned from input images into a latent vector space. To solve the language problem the author uses LSTM instead of RNN to solve the vanishing gradient problem. LSTM are great for remembering information for a longer period of time unlike RNNs. The language model only focusses on the user interface layout, various components and finally their relationships. Their experiment results present an interesting incentive to use deep learning models over traditional AI and ML techniques due to using unsupervised learning which does not need any human labelled data.

The objective of this semesters research was directed towards deep learning techniques to improve interface generative/reconstruction using feature extraction with Artificial Intelligence models. Deliverable 1 will cover Google's Tensorflow implementation of Drawing Classification while Deliverable 2 and 3 explore Shape and component classification using Doodle Classifier. Deliverable 4 describes a web server to receive component classifications with information about width and height of bounding boxes and other meta information.

II. Deliverable 1: Tensorflow implementation of Drawing Classification

This deliverable consisted of researching and implementing Google's Tensorflow implementation of Recurrent Neural Networks for Drawing Classification. The recognition is performed by a classifier that takes the user input, given as a sequence of strokes of points in x and y, and recognizes the object category that the user tried to draw.

The model uses a combination of convolutional layers, LSTM layers, and a softmax output layer as shown below to classify the drawings:



The input is a drawing that is encoded as a sequence of strokes of points in x, y, and n, where n indicates whether a the point is the first point in a new stroke.Then, a series of 1-dimensional convolutions is applied. Then LSTM layers are applied and the sum of the outputs of all LSTM steps is fed into a SoftMax layer to make a classification decision among the classes of drawings that we know.

Run the code:

1) Install Tensorflow
2) Download the deliverable 1 code
3) Execute the code with the following command to train the RNN-based model.

```
python train_model.py \
    --training_data=rnn_tutorial_data/training.tfrecord-?????-of-????? \
    --eval_data=rnn_tutorial_data/eval.tfrecord-?????-of-????? \
    --classes_file=rnn_tutorial_data/training.tfrecord.classes
```

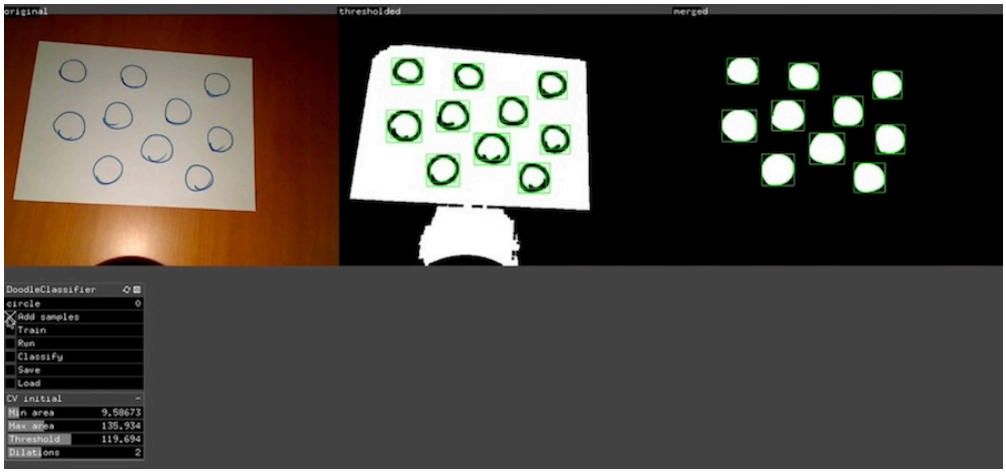
III. Deliverable 2: Hand drawn shape classification using Doodle Classifier

DoodleClassifier is an openFrameworks application, part of the ml4a-ofx collection, which lets you train a classifier to accurately recognize drawings ("doodles") from a camera. It was first used in a project called DoodleTunes by Andreas Refsgaard and Gene Kogan, which used the app to recognize doodles of musical instruments and turn them into music being made in Ableton Live.

In this deliverable, we train doodle Classifier to learn to recognize and classifiy simple hand drawn shapes such as triangles, circles and stars.

DoodleClassifier is quick and easy to set up. It's also advised to use thick markers rather than pencils or pens, because the lines can be more easily distinguished by the software. Take 4 sheets of paper. On three of these sheets draw circles, triangles and stars. On the 4th sheet draw a mix of stars, circles and triangles. Before launching the app, review and adjust the settings, which can be found in the file settings_doodleclassifier.xml. In that file, you must define the classes you'd like for the app to recognize. By default, they are circle, star, and arrow, but you may change those and have as many different classes as you'd like.

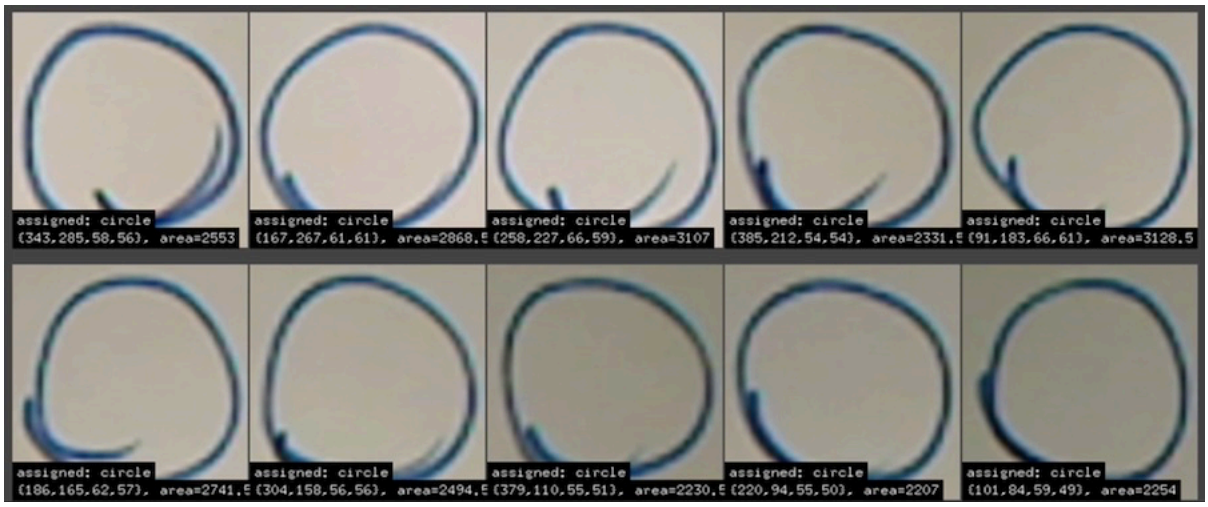
During training, you are giving the app as many examples of each class as you can. The first slider at the top of the GUI lets you select one of your defined classes as the active class. On a piece of paper, draw some instances of that class and put them underneath the camera. You will probably need to adust the computer vision settings to properly identify and segment them. A screenshot from drawing some instances of circles, and an explanation of what the CV parameters do is given below.



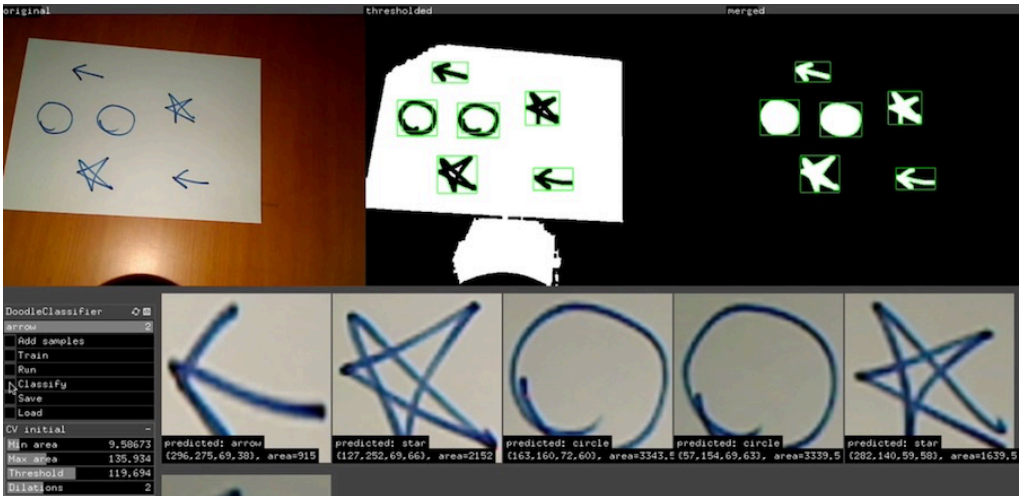
The CV parameters are found at the bottom of the GUI. The first slider you should adjust is the Threshold which determines the brightness threshold by which to separate the foreground from background content. Here you may have to adjust your physical setup to minimize the influence of shadows, which may interfere with this process. An overhead light may be helpful. The Dilations slider dilates (thickens) the discovered lines, and may help reduce fragmentation of found instances.

Finally, the Min area and Max area sliders control the acceptable range of sizes of drawn instances to allow. If you set Min area too low, you may have a lot of spurious doodles discovered. The ideal

is to have segmented your doodles (inside the green rectangles), but not anything else, which may corrupt your classifier. The screenshot above is an example of successful segmentation. When you have achieved this with your first class, click Add samples, and after a moment, during which a convnet analyzes the features of each doodle and saves the feature vector to memory, the samples will appear below the camera images. For example:



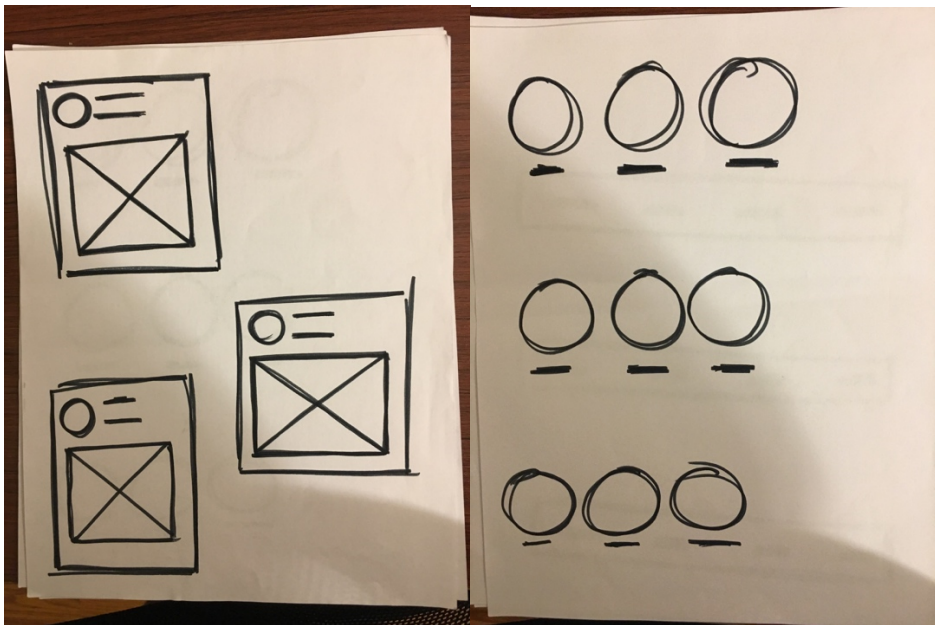
Now repeat this process with all of your classes, by moving the first slider in the GUI to each class in order, and drawing instances of that class underneath the camera, and clicking Add samples. Once you are ready, click Train in the interface, and wait for the training to complete. This may take anywhere from a few seconds to a few minutes depending on the complexity of your dataset. Once training has completed, you can classify new images. Draw some instances of your class, put them under the camera, and click Classify. After a moment, it will segment your doodles as before, and predict which classes they belong to. It will instantly send each predicted class as an OSC message to the address given in the settings, where the value of the OSC message is a string corresponding to the name of the predicted class, and four floats corresponding to the x, y, width, and height of the bounding box.

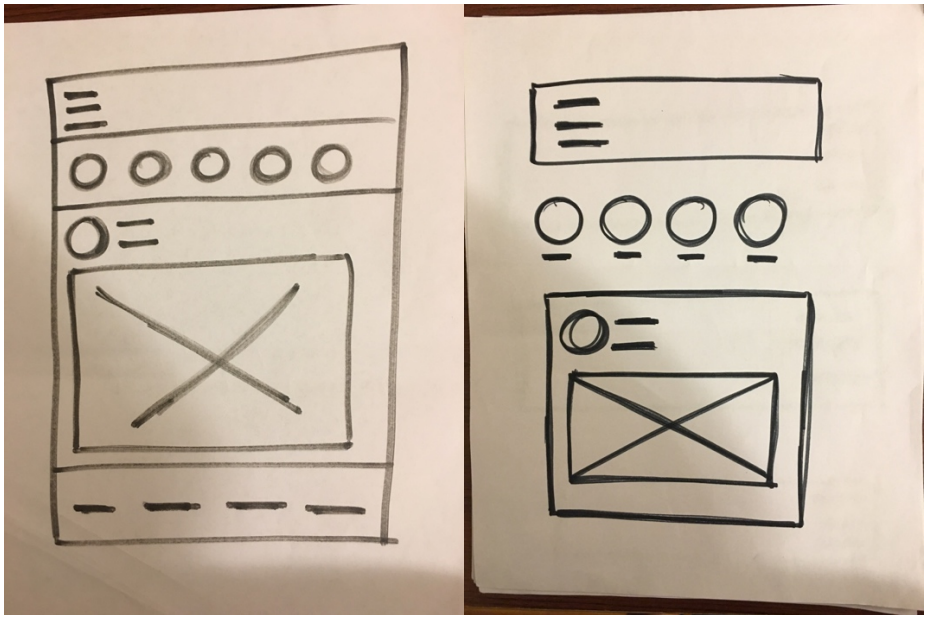


IV. Deliverable 3: UI Component Classification using Doodle Classifier Conclusion

In this deliverable we continue on the work developed in the previous deliverable. Here we train doodle classifier against UI components for classification.

We first define a set of components to train the classifier against. I took inspiration from the Instagram mobile app to define these components which were the ‘top-navigation’, ‘stories’, ‘image card’, ‘bottom-nav’. We draw each of these components on single sheets of paper using thick black markers. We then proceed to train the doodle classifier against each of the classes and repeat the process as defined in above deliverable.



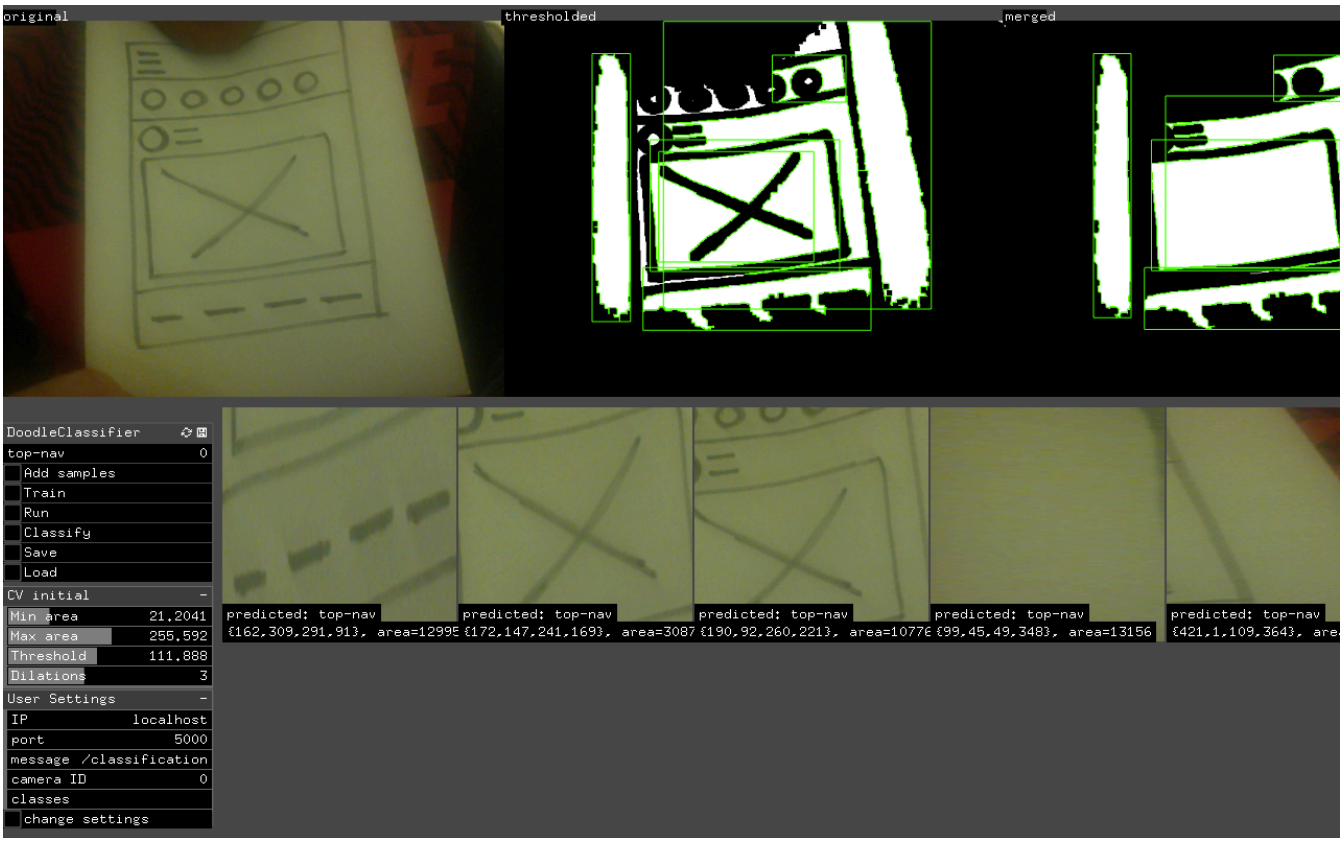


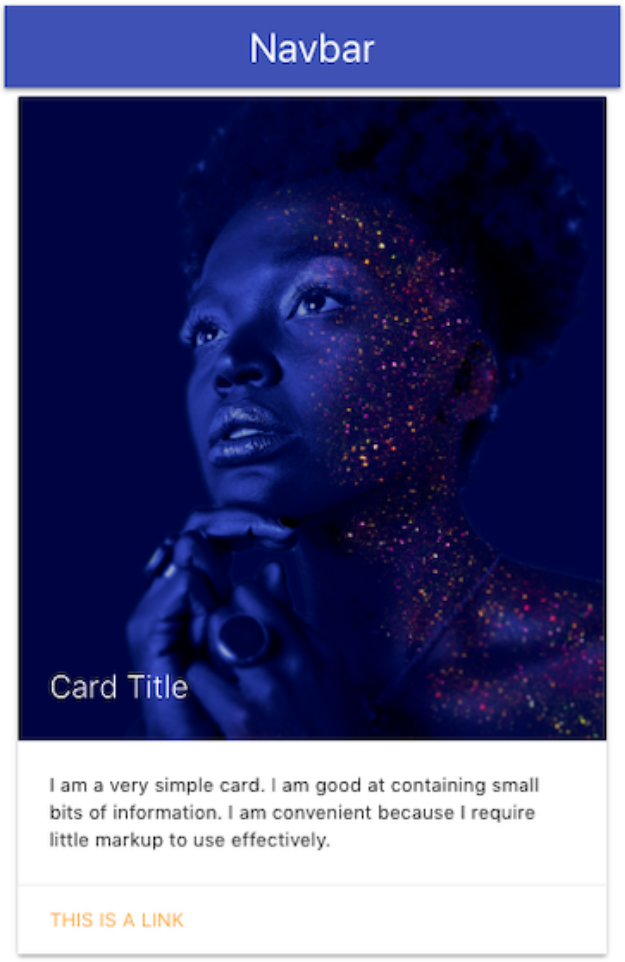
V. Deliverable 4: Web Server to convert and render Component Detections

In this deliverable, we build a web server to receive information regarding the detections made. The web server is built using NodeJS which is a JavaScript runtime built on Chrome's V8 JavaScript engine. The web server uses the OSC (open sound control) protocol to receive data from the doodle classifier. Open Sound Control (OSC) is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology. Bringing the benefits of modern networking technology to the world of electronic musical instruments, OSC's advantages include interoperability, accuracy, flexibility, and enhanced organization and documentation.

Once the classifier has been trained, the class and bounding box rectangle are sent over OSC (open sound control), to the IP address and port specified in the settings file, localhost:5000 by default, using the address \classification. You may change these accordingly, and will easily send to another computer over the same network if you change the IP address. The precise order of the info it sends for each detected object is: class (string), x-position (float), y-position (float), width of rectangle (float), and height of rectangle (float).

After receiving input from doodle classifier over OSC, the web server renders these detections on a web page to present to the user.





Classifications are rendered on a web page in real time.

VI. Conclusion

The CS297 report identifies and talks about how researchers and designers are developing numerous solutions for improving the user interface design process. Updating the UI design process has been long overdue and the recent advancements in AI and ML especially with the help of deep learning have been significant towards the research of this field. [1], [2] and [3] research about CNN's and how they can be used to capture visual features from images or videos. [1] goes on to mention the use of RNN or a combination of CNN and LSTM to work with a sequence of images instead of a static screenshot. Finally in [5], Vinyals et.al discuss generating text captions from images and how they can be used to possible generate descriptions of user interface components to reconstruct entire interfaces.

Generating computer code from a graphical user interface input in the form of an image is a very new research topic. Advancements in AI and ML will only propel this research more. The only drawback found was the lack of training data. AI algorithms require data in the order of millions to be able to reach general accuracy. [1] discusses training these algorithms without the need for human labelled data which can be researched more since this can help cut down training and application deployment significantly.

In CS298, I will be working on improving UI component detection by fine tuning various hyperparameters followed by building up on Deliverable 4 to develop a refined and easy-to-use end-to-end AI assisted UI development pipeline.

References

[1] Beltramelli, T. (2018). "*pix2code: Generating Code from a Graphical User Interface Screenshot*." [online] Arxiv.org. Available at: https://arxiv.org/abs/1705.07962 [Accessed 2 Nov. 2018].

[2] O'Shea, K. and Nash, R. (2018). "*An Introduction to Convolutional Neural Networks*." [online] Arxiv.org. Available at: https://arxiv.org/abs/1511.08458 [Accessed 2 Nov. 2018].

[3] Liu, X., Chen, Q., Wu, X., Liu, Y. and Liu, Y. (2018). *CNN based music emotion classification*. [online] Arxiv.org. Available at: https://arxiv.org/abs/1704.05665 [Accessed 2 Nov. 2018].

[4] Donahue, J., Hendricks, L., Rohrbach, M., Venugopalan, S., Guadarrama, S., Saenko, K. and Darrell, T. (2018). "*Long-term Recurrent Convolutional Networks for Visual Recognition and Description*." [online] Arxiv.org. Available at: https://arxiv.org/abs/1411.4389 [Accessed 2 Nov. 2018].

[5] Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). "*Show and tell: A neural image caption generator.*" *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3156-3164.

[6] Mankar, Vijay & G Bhele, Sujata. (2012). "*A Review Paper on Face Recognition Techniques.*" International Journal of Advanced Research in Computer Engineering & Technology. 1. 339-346.

[7] S. Shwartz and S. David, "Understanding machine learning from theory to algorithms", publication date: February 2014. [online] Available: http://www.cs.huji.ac.il/~shais/understanding-machine-learning-theory-algorithms.pdf.

[8] A. Tuguai and R.Xing, "Reflections on the limits of artificial intelligence", in Ubiquity, New York, NY, USA, article 3, pages 2, publication date: December 2004, DOI: 10.1145/1041062.1041064

[9] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. Deepcoder: Learning to write programs. arXiv preprint arXiv:1611.01989, 2016.

[10] A. L. Gaunt, M. Brockschmidt, R. Singh, N. Kushman, P. Kohli, J. Taylor, and D. Tarlow. Terpret: A probabilistic programming language for program induction. arXiv preprint arXiv:1608.04428, 2016.

[11] W. Ling, E. Grefenstette, K. M. Hermann, T. Kocisk ˇ y, A. Senior, F. Wang, and P. Blunsom. ` Latent predictor networks for code generation. arXiv preprint arXiv:1603.06744, 2016.