YODA – YOUR ONLY DESIGN ASSISTANT

A Project Presented to

The Faculty of Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Siddharth Kulkarni

May, 2019

The Designated Project Committee Approves the Master's Project Titled

YODA – Your Only Design Assistant

By

Siddharth Kulkarni

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2019

Dr. Christopher Pollett	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Mr. Kevin Smith	Department of Computer Science

ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude to Dr. Christopher Pollett for his pertinent guidance, advice, and collaboration throughout the duration of my project. I consider myself to be extremely fortunate to have had an opportunity to work with someone as brilliant as him.

I am also grateful to my committee members Dr. Robert Chun and Mr. Kevin Smith for providing their valuable feedback and guidance.

Finally, I thank my wonderful parents and friends for their countless hours of support and encouragement along the way.

ABSTRACT

YODA – Your Only Design Assistant

By Siddharth Kulkarni

Converting user interface designs created by graphic designers into computer code is a typical job of a front end engineer in order to develop functional web and mobile applications. This conversion process can often be extremely tedious, slow and prone to human error. In this project, deep learning based object detection along with optical character recognition is used to generate platform ready prototypes directly from design sketches. Also, a new design language is introduced to facilitate expressive prototyping and allowing the creation of more expressive and functional designs. It is observed that the AI powered application along with modern web technology can significantly help streamline and automate the overall product development routine and eliminate hurdles from the product development process.

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

TABLE OF CONTENTS

CHAPTER

INTRODUCTION

The user interface of an application involves the components that a user is presented with and interacts with directly on their screen. It includes the basic components needed for a user to navigate across products and complete user tasks. Common UI components include: navigational and input controls, information containers, interactive elements such as buttons, toggles, dropdown and radio buttons. All successful products share significantly easy to user interfaces and are the primary contributor of incoming traffic and product usage. People are surprised by the impact the user interface of a product can have on its success or failure. It is the reason why almost all product companies today invest heavily in hiring designers and engineers to study, develop and improve their product interfaces.

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.

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 researchers from the Google Brain team were able to train a machine to produce captions to describe images. The algorithm which used the Inception v2 image classification model with an accuracy of 98% 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. 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 research project was to develop deep learning techniques to improve interface generative/reconstruction using feature extraction with Artificial Intelligence models. Various object detection models and techniques were studied and finally the proposed solution consisting of a combination of object detection along with optical character recognition was used to achieve the artificially intelligent application. The object detector powered by YOLO is able to achieve real-time detection at 45 frames per second [16] making it suitable for use in an industry standard design assistant application.

This report is organized in five different chapters. The first chapter deals with the problem statement, related work and contribution of this research project. The second chapter discusses the background needed to understand and implement the proposed solution. The third chapter talks in detail about the design and implementation process of the proposed solution followed by the Experimental Results in the fourth chapter. Finally, Chapter 5 concludes the report along with future work and direction.

BACKGROUND

This section discusses the background needed to understand and implement the User Interface Design Assistant powered by deep learning. This section has been divided into two sub sections – design and engineering. The first sub-section explores the topics of wireframes, prototypes, mockups and UI design systems. The engineering sub-section covers Neural Networks, convolutional neural networks, and the deep learning library darknet and optical character recognition with Tesseract.

DESIGN

Wireframes

A design wireframe is a visual blueprint that represents the skeletal framework of a product. They are used to best accomplish a particular purpose of a product by arranging elements with a business objective yet remaining creative. The wireframe describes product layout w.r.t user interface elements, navigation and how various elements interact with each other. The primary focus of wireframes is in product functionality, priority of content and behavior and hence lacks emphasis on typography and any higher form of graphics. Wireframes are usually simple pen and paper based drawings or sketches. Teams of designers collaborate using designs on whiteboards and various commercial software applications available. Wireframes are created and used by many users across teams and platforms such as designers, developers, product managers and business analysts.

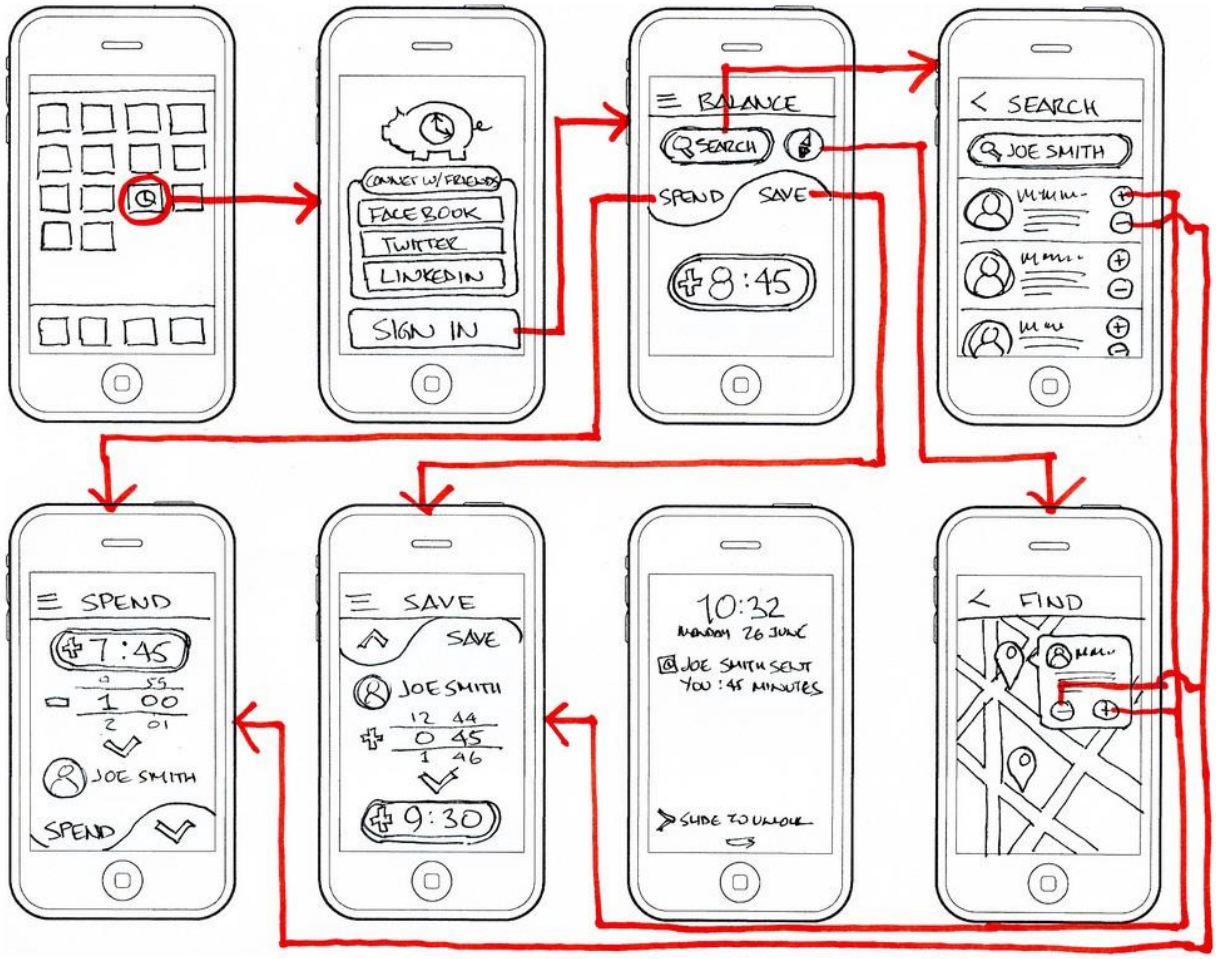


Fig.1 Modern Mobile Prototypes

Prototypes

Design prototypes have a different set of requirements as compared to wireframes. The primary requirements being they must be of high fidelity, be interactive and as close in fit to the final user interface as possible. They are usually tested against a set of test user groups and can help save a lot of development time and money. Prototypes involve no HTML/CSS/JS and no considerations of server and database are made till this stage. Prototypes are often where designers share their work with developers for the first time and begin user interface discussions and back-end

feasibility. There are various software tools which enable prototyping such as Mockplus, iDoc and UXPin.

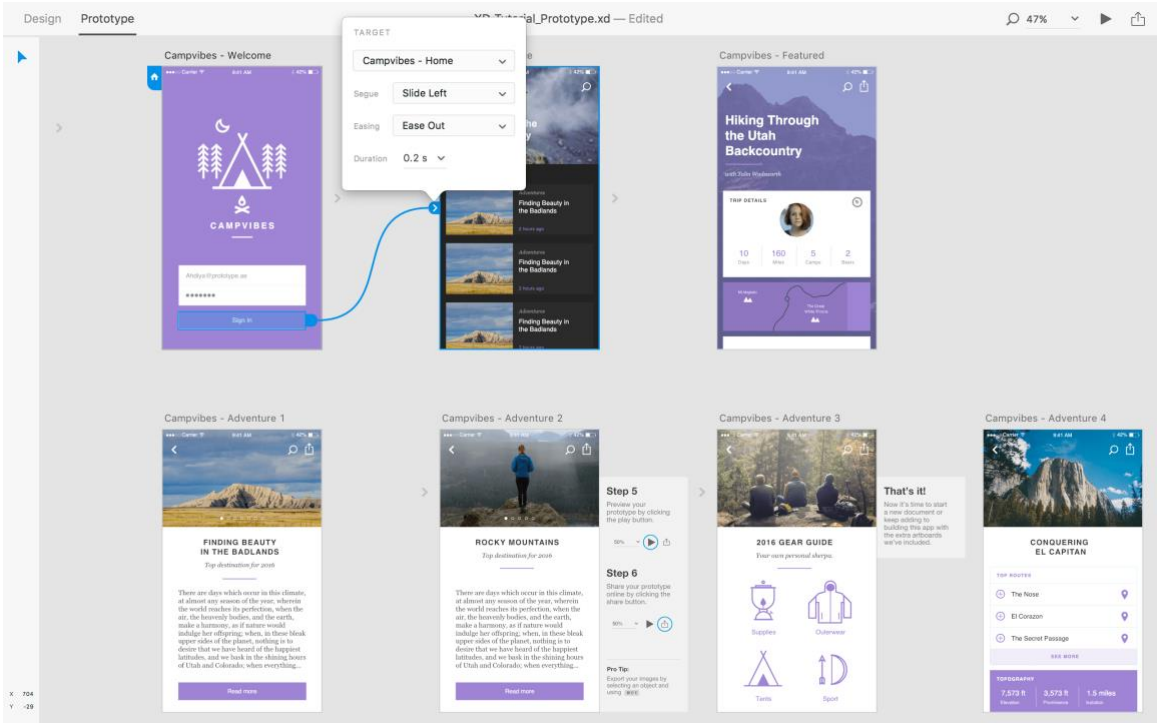


Fig.2 Mockup Example from [13]

Mockups

Mockups are used for the overall visual design of a product and can be considered as a "visual script". They usually have a far more richer visual and descriptive graphics than wireframes such as layout, color and visual presentation. Unlike prototypes, mockups are often static assets with focus on the appearance of the product rather than interaction. That often used for getting quick feedback and other improvements to the visual design.

Fig.3 Fan made iOS UI Design System [13]

UI design systems

When companies become big enough where there are multiple design and front end teams across the company, UI design systems are used to enforce a common set of components which help manage design at scale. UI design system can apply to all digital products used by the company such as websites, ads, mobile applications, etc. There are various titles which come under the UI design sytem umbrella such as Pattern library, Modular design, Component design, Design language, User interface library. UI design systems can also be applied as a brand style and guide. There are various advantages to using UI design systems, such as:

- Consistency
- Scale
- Cross Team Collaboration
- Efficiency

Neural Networks

Neural Networks are a subfield of machine learning where algorithms modelled after the human brain are researched and developed. The human brain can be considered as a set of neurons interconnected to each other. Neurons are the basic building block and working unit of the nervous system. They are designed to transmit data or information received from other neurons to other neurons, muscles and glands. Their only function is to receive process and transmit information. These neurons receive input from another neuron or layer of neurons, process and forward these signals to other neurons. They are designed to recognize patterns by interpreting a combination of sensory data as input. They can be considered as models which help cluster and classify data which can be in the form of images, text, sound, etc converted or translated into numbers contained in vectors.

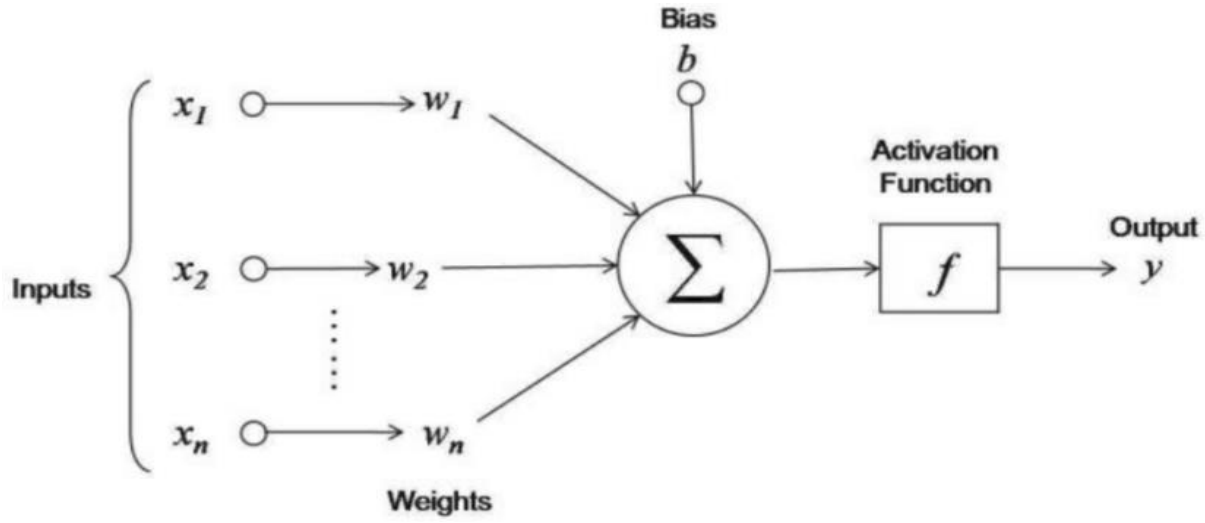


Fig.4 Neuron – basic unit of a Neural Network[14]

A node similar to a neuron is where some form of computation happens upon encountering significant stimuli. A node takes into consideration input data from previous nodes along with a

set of weights which determine whether that signal needs to be amplified or dampened. As shown in Fig.4 input-weight products are summed and an activation function is applied to determine the extent to which signal must progress further. As shown in Fig.5 , this neural network has five layers and each node in a layer is connected to all nodes in the previous and next layer.

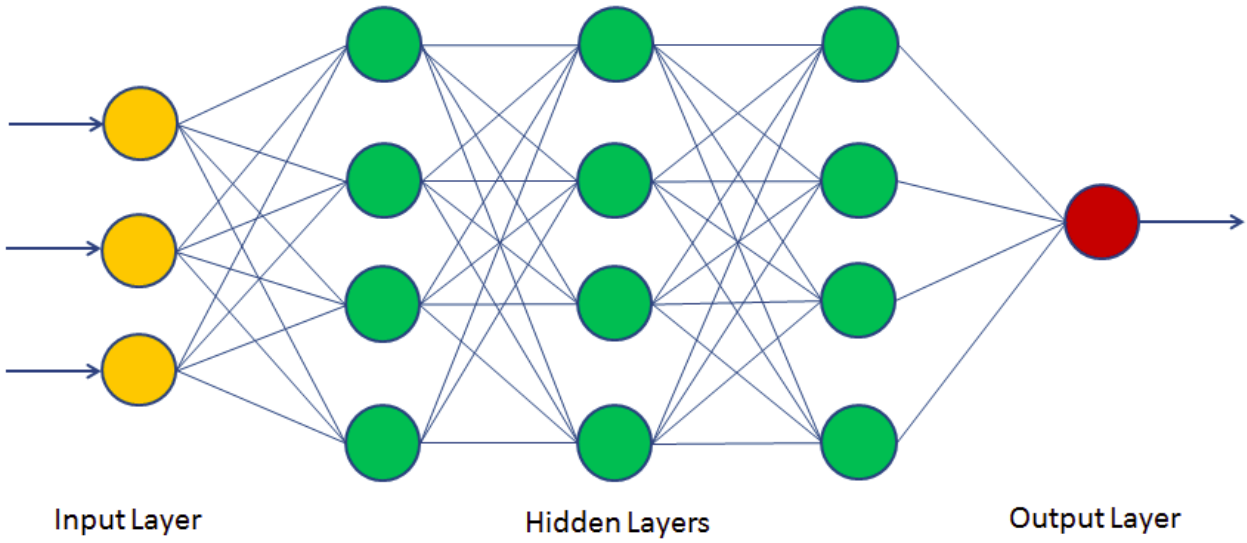


Fig.5 Example of a Neural Network [14]

Some of the applications of Neural Networks are:

- Character Recognition
- Image Compression
- Stock Market Prediction
- Traveling Salesman Problem
- Autonomous Vehicles

Object Detection with Convolutional Neural Networks

The concept of Convolutional Neural Networks (CNN) comes from the field of digital signal processing where two signals are convoluted to form a resulting signal. In deep learning, convolution operations are used to mix information from multiple sources to get a desired output. The convolutional mathematical operation is the fundamental operation by which convolutional neural networks work. That is the reason why CNNs are used most popularly for computer vision tasks as they are close to how the human brain tries to understand and make sense of our vision and perception of our surroundings. Other applications are video analysis, natural language processing, health risk assessment, biomarkers of aging discovery and more.

CNNs use a small window called a kernel or feature map to focus on a certain part of the input at a given time. This same window is used across the entire input data to identify various features and finally a pooling operation is applied on the convoluted output to get the extract the most important features.

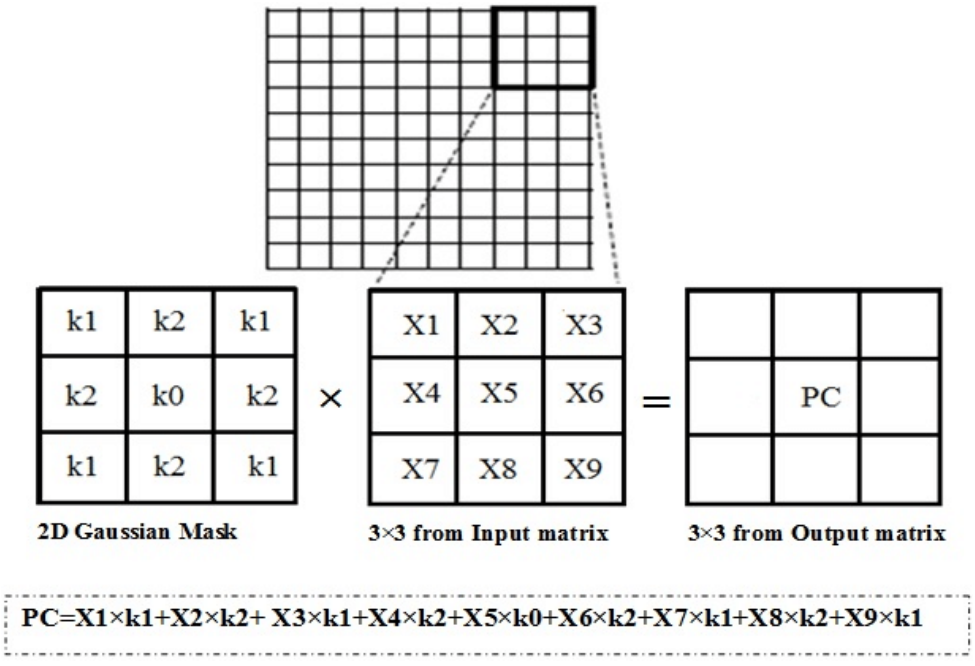


Fig.6 Kernel Operation on Input Image [14]

As shown in Fig. 7 , the kernel is mapped across the input and a dot operation of the imposed image and kernel is applied. The same kernel now slides across the entire image and finally pooling operation is applied. Pooling layers are used to reduce the dimensionality of the data by combining the outputs of neuron clusters in a particular layer as a single neuron for the next layer.

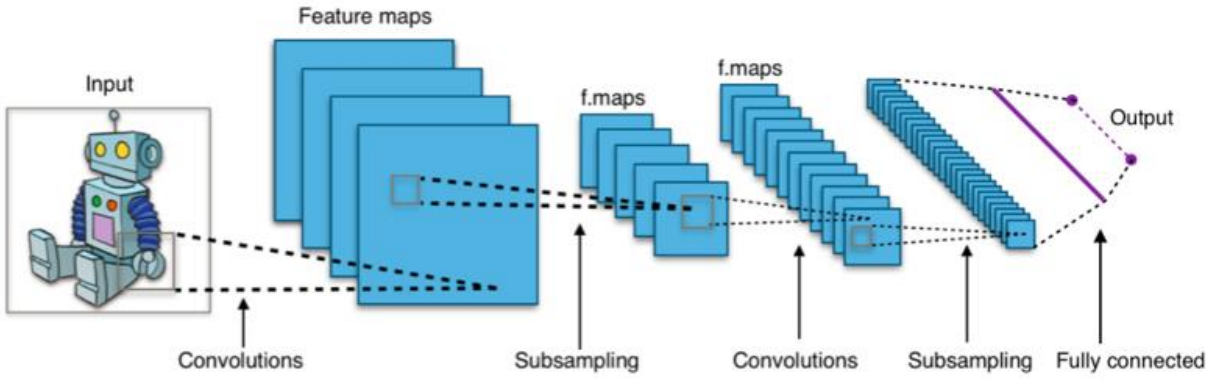


Fig.7 Convolutional Operation on sample Image [14]

Optical Character Recognition

Optical Character Recognition abbreviated as (OCR), is the conversion of images of printed, typewritten or handwritten text into digital machine-encoded text. It is often used for converting a physical source of truth such as a legal document, receipts, mail into a digital format for storing compactly, fast look ups and availability through the internet. OCR involves various fields such as computer vision, artificial intelligence and pattern recognition.

Images are first scanned to convert the physical documents into their digital representation. The scanned image is then analyzed for dark and light regions. The light regions are identified as background elements while darker regions as text to recognized. Further processing is applied onto the dark regions to classify them as digits or alphabets. There are various algorithms for identifying

characters and they can be classified into two either pattern recognition or feature detection. When a character is finally identified, it is converted into an ASCII code that is usable by computer systems.

There are various applications of OCR such as: scanning printed documents, automated data entry, archiving historical information, sorting letters for mail delivery, etc.

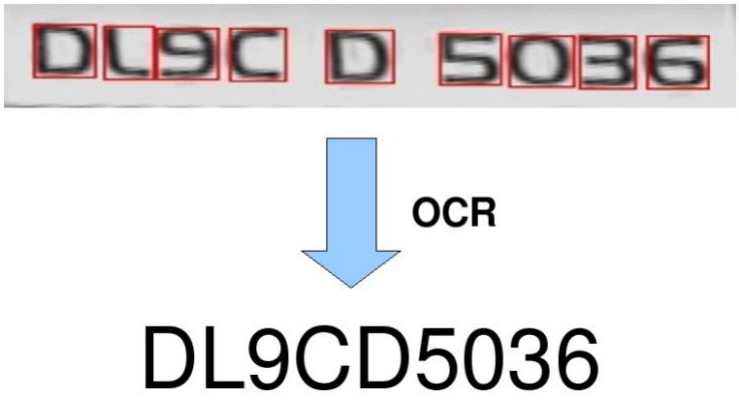


Fig.8 Example of OCR on Car License Place [15]

DESIGN AND IMPLEMENTATION

This section describes the new design prototyping language followed by the tools and libraries used for the experiments. The implementation can be divided depending on the technology stack they use primarily, user interface design language which is implemented on pen and paper, openframeworks which is a C++ wrapper for creative programming and finally the web stack consisting of HTML, CSS, JS and NodeJS.

Environment and Openframeworks

Openframeworks is an open source toolkit written in C++ on top of OpenGL with the purpose of "creative coding". It was founded by Zachary Lieberman, Theo Watson and Arturo Castro and is maintained by contributions from the openFrameworks public community. Openframeworks was built to work as a general purpose tool and wraps around various other open source libraries such as GStreamer, OpenCV, Quicktime, Assimp, cairo and more. Openframeworks is massively cross-platform with support for Linux, OSX, iOS, Windows, Android and the IDEs Visual Studio, xCode, Eclipse, Code::blocks and more.

Additionally, developers have made available various plugins, extensions and libraries. An addon can be considered as code which extends openFrameworks functionality in some way or the other and make a complicated task simple and reusable for other programmers. Addons have been made for achieving tasks such as graphics, GUI, algorithms, animation, computer vision, machine learning, physics, sound, etc which can be found on ofxaddons.com.

Doodle Classifier

Doodle Classifier is an openFrameworks application which lets users train and classify a model to accurately recognize drawings or ‘doodles’ from a camera. It was originally used in a project called DoodleTunes by Gene Kogan and Andreas Refsgaard to classify hand draw musical instruments and generate music using Ableton Live. It was inspired by the research made by Jonas Jongejan et al. at the Google Creative Lab working on the QuickDraw app.

Fig.9 Doodle Classifier Setup

The setup is fairly simple and straight forward. A camera device is required, and works best when an overhead camera pointing downwards on a preferably white or lighter shade table is used. Also preferably a thick black marker should be used for the drawings as it helps distinguish elements better for the software.

As the application starts, you can edit and add your own class names which the classifier should recognize. After the application has been trained, the classification and bounding box coordinates within the image are sent over Open Sound Control (OSC) to the IP address and port of your choosing. OSC is a protocol which enables encoding for multimedia applications such as communication among sound synthesizers, computers and other multimedia devices.

Once a list of classes have been defined, the classifier now needs to be trained with as many examples of each class. More examples ensure higher accuracy rate of the classifier. As input, draw a few instances of the class you want to train on a piece of paper and place them under the camera. A couple of computer vision settings may need to be changed depending on the light settings of the room to ensure proper segmentation and identification. Fig.10 below is a screenshot from training the model to learn circles.

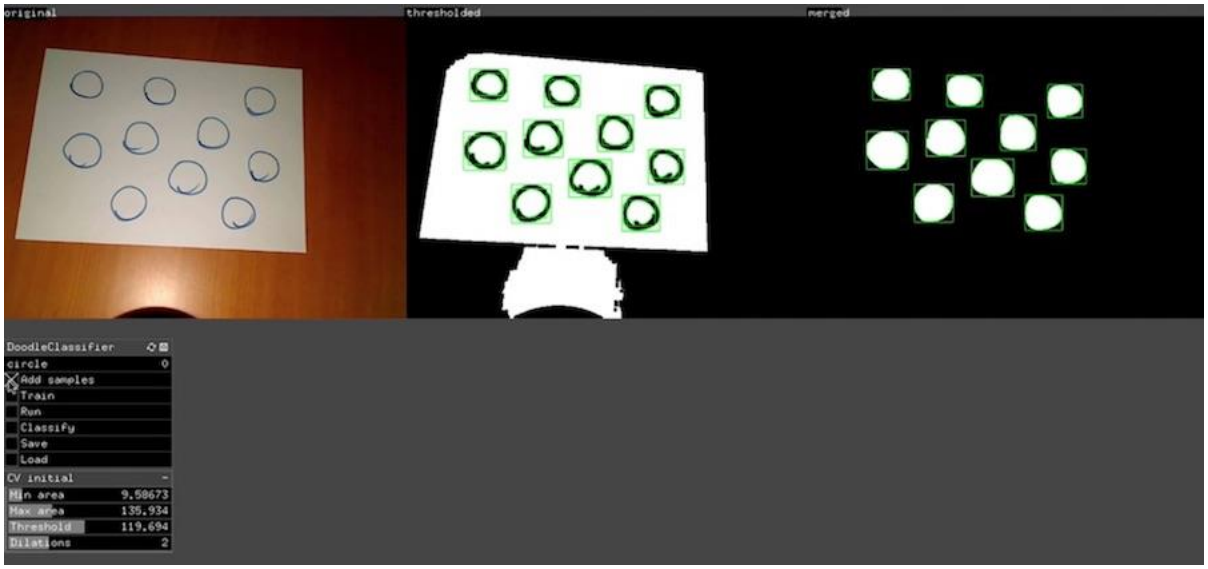


Fig.10 Training classifier to learn Circles

The various CV parameters that need to be adjusted are:

- **Threshold**: this determines the brightness threshold which is used to separate the foreground and background
- **Dilations**: this helps to dilate(thicken/thin) the discovered lines helping reducing fragmentation of shadows.
- **Min and Max area**: these values can be adjusted to control the acceptable area for the application to allow for training or classification.

After successful segmentation, the convolutional neural network analyzes the features of each doodle or drawing and saves the feature vector to memory. The images are displayed as below.

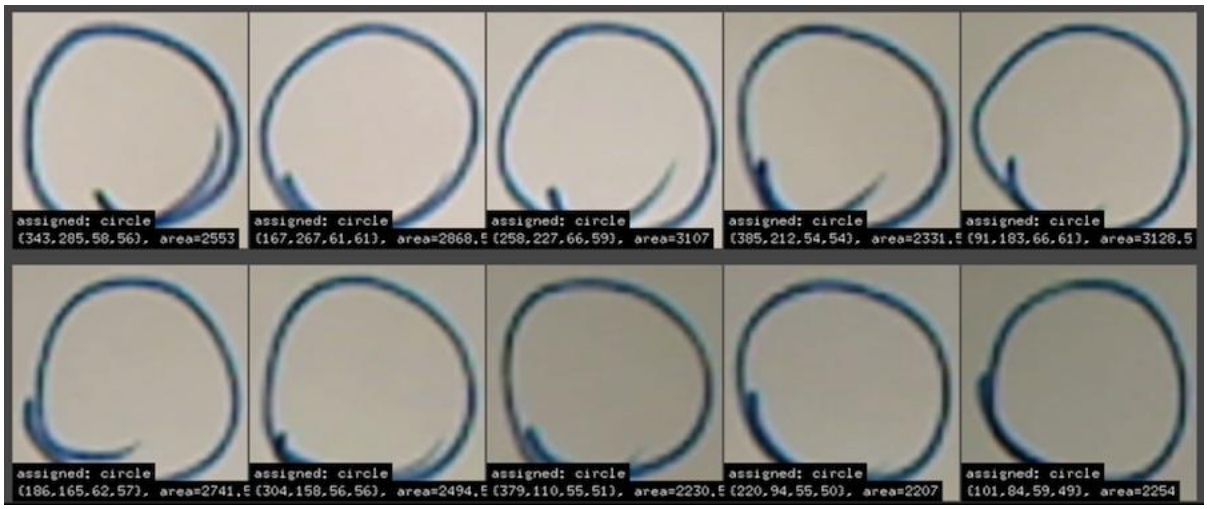


Fig.11 Classifier trained on Circles

Now this process will be repeated for all classes that the classifier needs to be trained on. An interesting question which rises is how many instances of each class need to be drawn and used for training. The answer being it depends heavily on the properties and quality of the image

instances. Listed below are the characteristics which make classification harder and more complicated:

- Increasing number of distinct classes
- Variance between image instances in terms of design and aesthetic
- Similarity index between two images i.e.. more similarity between two classes confuses the classifier even more.

In our experiments, providing 5-10 samples of each class were sufficient to properly train the classifier. Now once training has been completed, provide new instances of a classification and let the application classify the image. After a while, the doodles will be segmented and class prediction made. All of the predictions along with important positional meta data is sent over the address in the form of a OSC message using the OSC communication protocol. Each OSC message consists of the name of the predicted class and four floats corresponding to the bounding box coordinates within the image.

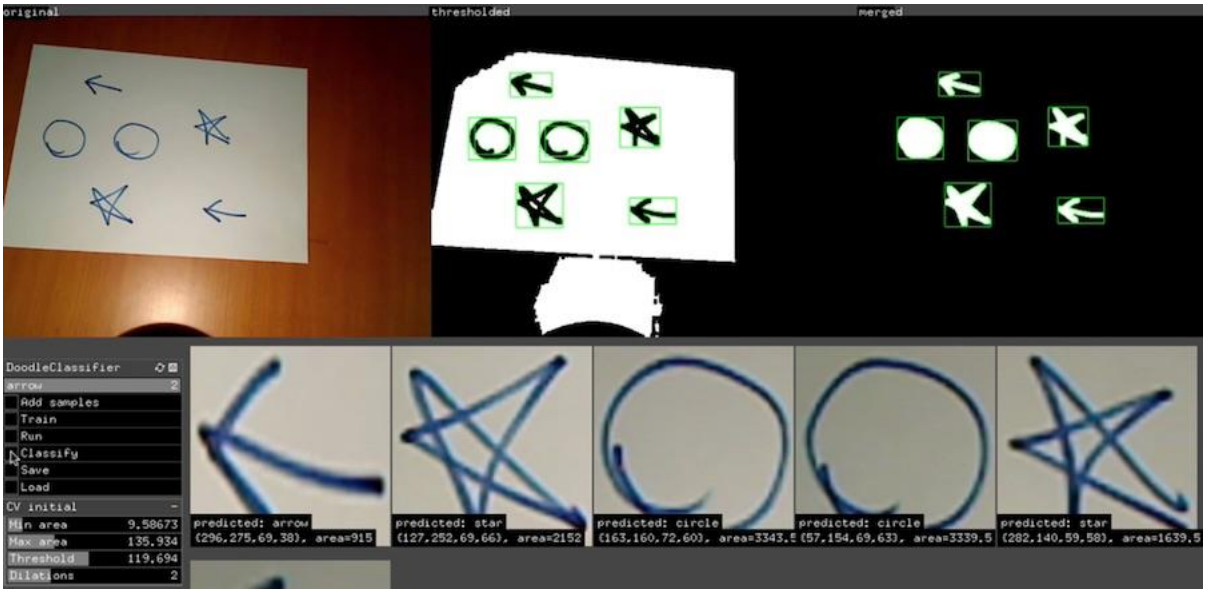


Fig.12 Classifier Predicting Circles and Stars

For the purpose of YODA, a set of UI components need to be used to train the classifier. The UI components from the popular photo-sharing app Instagram were used as inspiration to build our needed component library.

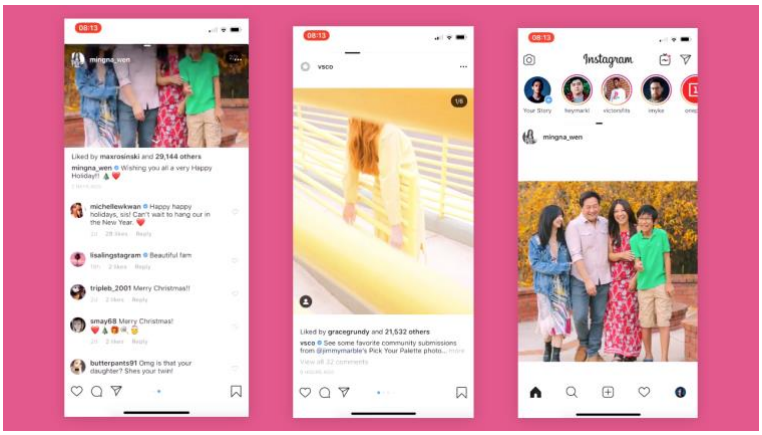


Fig.13 Instagram Mobile App Screenshots

The components decided for the YODA are as follows:

- Top-nav
- Image-card
- Bottom-nav
- Search-bar
- Profile-about
- Image-grid
- Activity
- Message-list

For each of these components, a simplified wireframe was made using a thick black sketch pen on a white piece of paper. It was seen that 4 instances of each component were good enough for

significantly distinct components. Components such as the top-nav, bottom-nav, search-bar, etc needed more than 8 instances to properly train the classifier.

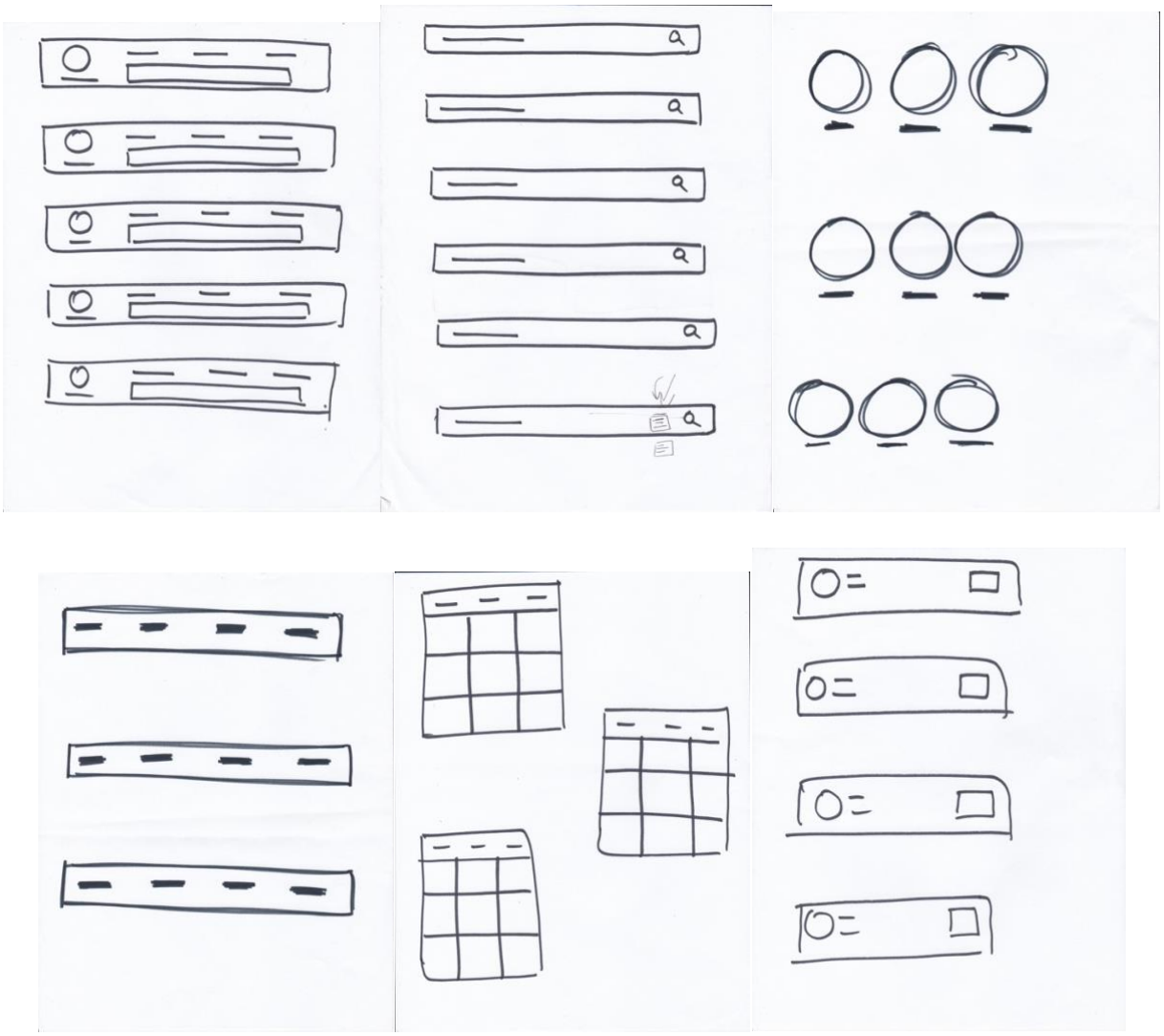


Fig.14 Screenshot of sample UI Components

Proposed Design Language

Now that a simple UI component detection system has been setup, we are restricted with a set of components with inflexible properties. These components have pre-defined properties such as style, interactive user actions and can only be altered by manually making changes to the component code.

A new prototyping design system is proposed which allows users to attach properties and build more complex components. Users can use predefined decorator followed by property definition. This enables users to still be able to rapidly design prototypes while abstracting away the technical complexity of user interfaces and focus more on the creative aspect of prototyping.

Before the types of properties are described, it is important to mention how these properties can be added to existing UI wireframes. Designs usually contain simple shapes such as boxes, circles, simple lines and sometimes text to denote components. Each component can be assigned different properties; this means that a designer needs to embed these properties within the design inside each of these components. In this way, the detector knows which component to assign what properties to. For this prototyping system, we have decided that the top-right space inside a component is reserved for component level properties. The space right outside the main parent component is for parent/global level properties.

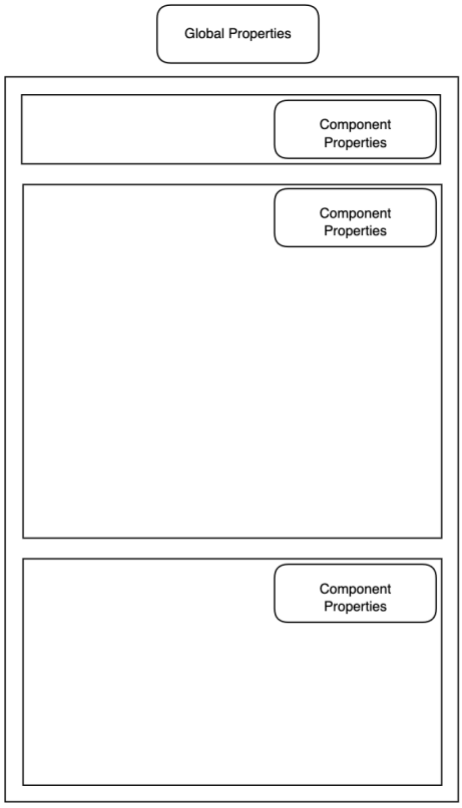


Fig.15 Global vs Component Level Properties

There are various possible properties that can be attached. Each of these properties can have a pre-defined decorator which is a special symbol denoting property type followed by property value. Multiple property values for the same decorator can be separated by a comma. They are listed below:

- Style ($): stylistic properties which affect border, text, background, margin, padding, etc. We can further pre-define sub-property decorators for each stylistic property value. Eg: BO for border, BA for background, C for color
- Actions (#): event driven properties such as trigger a function, browser alert, interface navigation, etc. Example usecase is button press on page @1 which navigates to page @2.

- Page definition (@): A subset of actions but more important for UI prototyping, page definitions can denote a reference to a component state.
- User defined custom properties: Users can make their own decorators with meaning for their purposes. Eg: Annotation for component library stack used like react, angular or jquery.

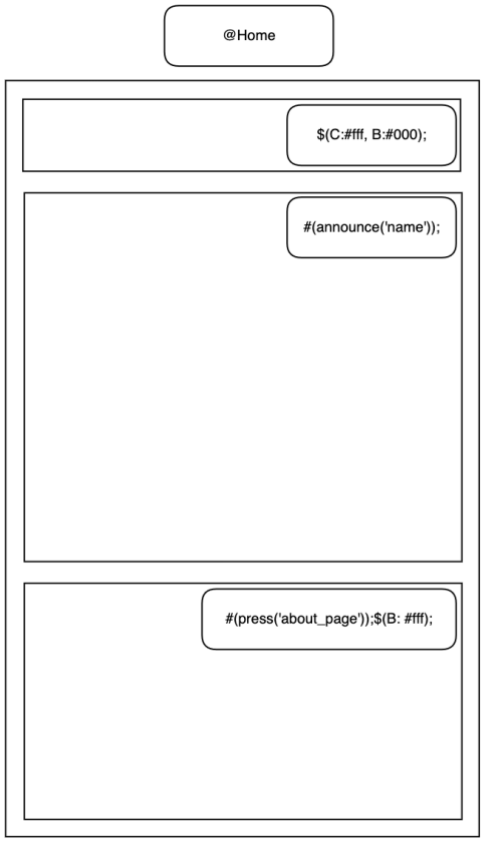


Fig.16 Example usage of property decorators

Tesseract

Tesseract is an open source project sponsored by Google for accomplishing simple Optical Character Recognition tasks with a significant success rate. Its OCR engine has support for over 100 languages out of the box and can be trained to recognize other languages as well. Google uses the Tesseract OCR engine in its Gmail image spam detection algorithm, text-detection on mobile devices and video applications.

For the purpose of YODA, a openFrameworks based wrapper ofxTesseract3 written by Wataru Kani is used for the Tesseract OCR engine. The installation steps and procedures can be found on my blog [12]. Tesseract OCR is applied on each detection sub-image and string result is attached to the meta-data in the OSC message and send to the web stack discussed ahead for further processing and rendering.

```
#include "ofxTesseract.h"
...
ofxTesseract ocr;
ofImage img;
...
ocr.setup();
ocr.setWhitelist("0123456789");
tess.setAccuracy(ofxTesseract::ACCURATE);
img.loadImage("text.png");
string result = ocr.findText(img);
cout << result << endl;
```

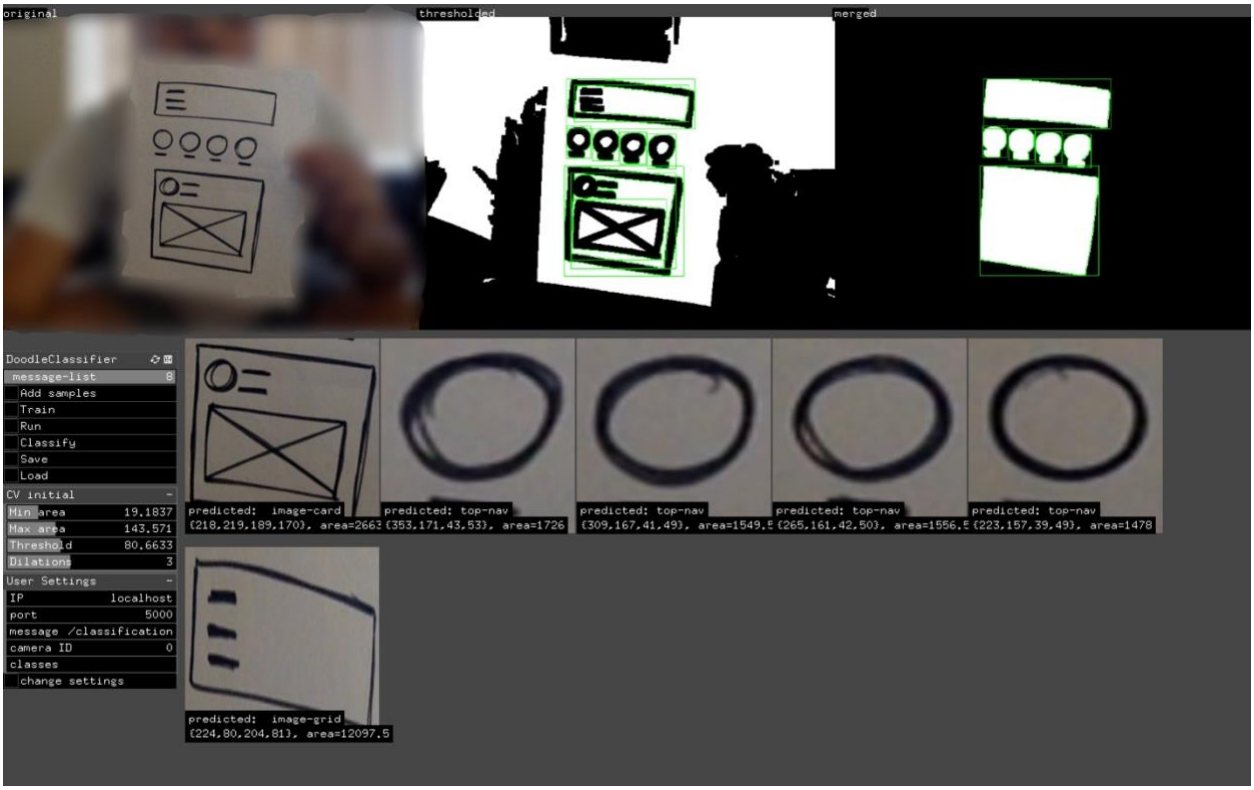
Web Stack

Once the classifier and Tesseract OCR engine have completed their task, all of component detections along with the required meta data is sent to the web stack of the project for live processing and rendering.

A simple NodeJS web server is setup which listens and captures OSC messages sent from the openFrameworks application. Here the basic templates of components are read from the database to create simple skeleton structures of the components. Next, component nesting is determined based on the bounding box coordinates sent as meta-data. In modern UI design, usually components are built from a set of smaller components and this way nested components can be determined. Finally, the OCR detected text is captured and first scanned for decorators. These decorators are useful for categorizing properties such as styling, event handling and more. Once these decorators have been identified, the various properties are applied to the component. Also these templates can be stored on a database such as MongoDB for later review.

Now that the properties have been applied and code has been formed for these components, they can be rendered live so that users can see what the results and what they are uploading.

For the purpose of live rendering, web sockets are used to communicate with a simple JavaScript front end application. Here users can accept or reject the rendered UI design. Additionally, users can see posts made by other users and add comments, likes and more similar to common photo-sharing apps like Instagram and Facebook.



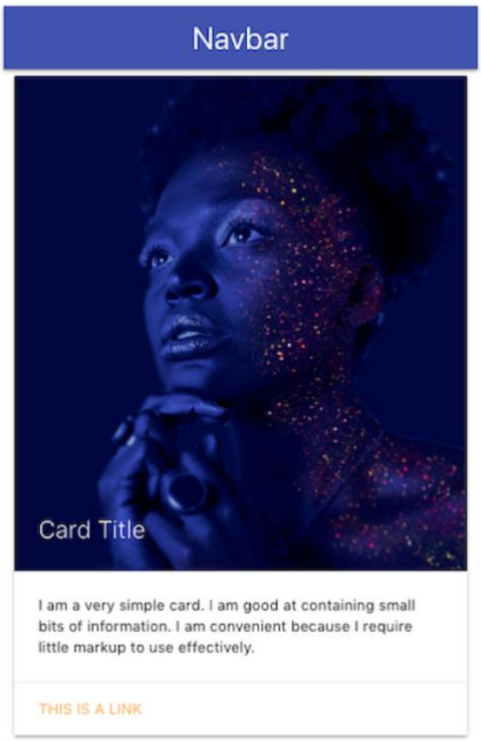


Fig.17 Live rendering design to web page

EXPERIMENTAL RESULTS

The experiments were conducted with a set of around 10 UI components. A black sketch-pen was used on white pieces of paper and there was an even distribution of similar and distinct looking components. The Computer Vision settings were fixed as follows:

- Min Area – 20.699
- Max Area – 126.132
- Threshold – 68.9541
- Dilations – 2

Experiments with Simple Components:

A search bar component is selected since this has a very simple structure consisting of a large rectangle, a line and search icon. As show in Fig.18, the classifier was able to classify all instances of the component successfully.

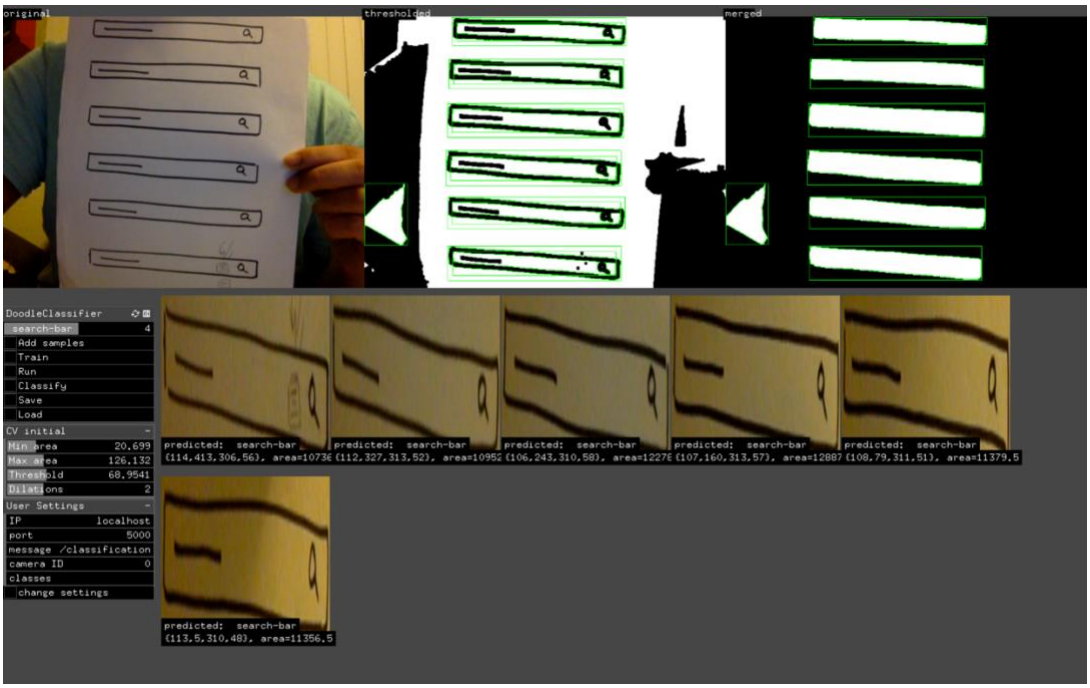


Fig.18 Predicting Search-Bar component

Experiments with Complex Components:

A profile bar component is selected for this example. As show in Fig.19, though the component looks very similar to the search-bar component, it has more complex features embedded such as a rectangle within a rectangle, a circle with a line and three more distinct lines. Again here, the classifier is able to successfully detect and recognize each instances of the drawing and provide the bounding box coordinates for each detection.

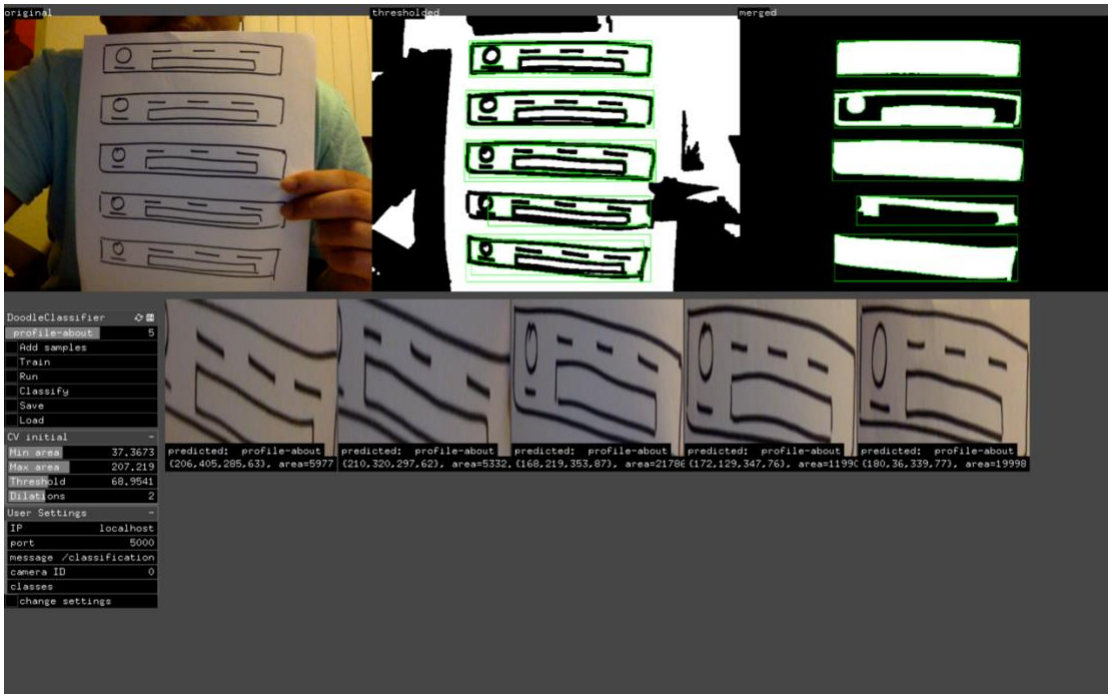


Fig.19 Predicting Profile-bar component

Note: the computer vision settings have been untouched and are the same used throughout all experiments.

Experiments with Similar Components:

In these experiments, similar looking components are showed to the classifier. The search-bar and profile-bar experiments are shown in Fig.20. The classifier is able to successfully predict component class and bounding box coordinates.

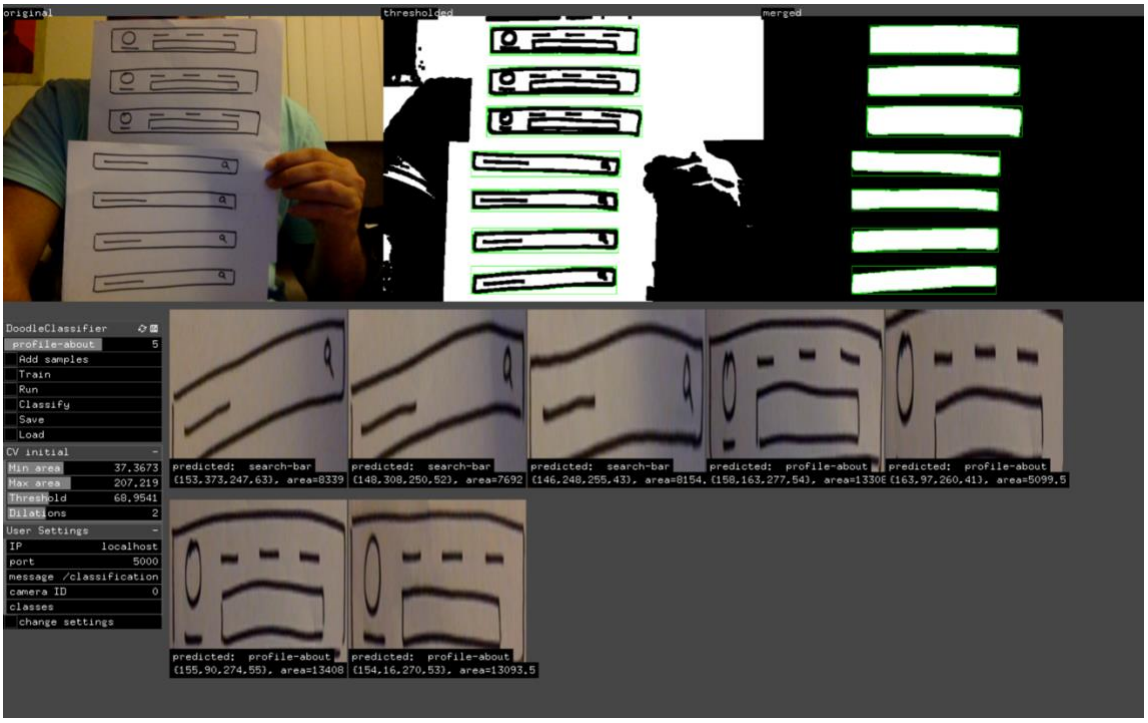


Fig.20 Predicting similar looking components

Note how the basic container or reactangle in this case is of near similar size. Apart from the profile-bar having a circle component and distinct search icon both contain rectangles and dashes.

Experiments with Distinct Components:

For these experiments components with significant distinct features were used. As shown in Fig.21, the image-grid component along with search-bar component are chosen due to difference in size, structure and basic component complexity.

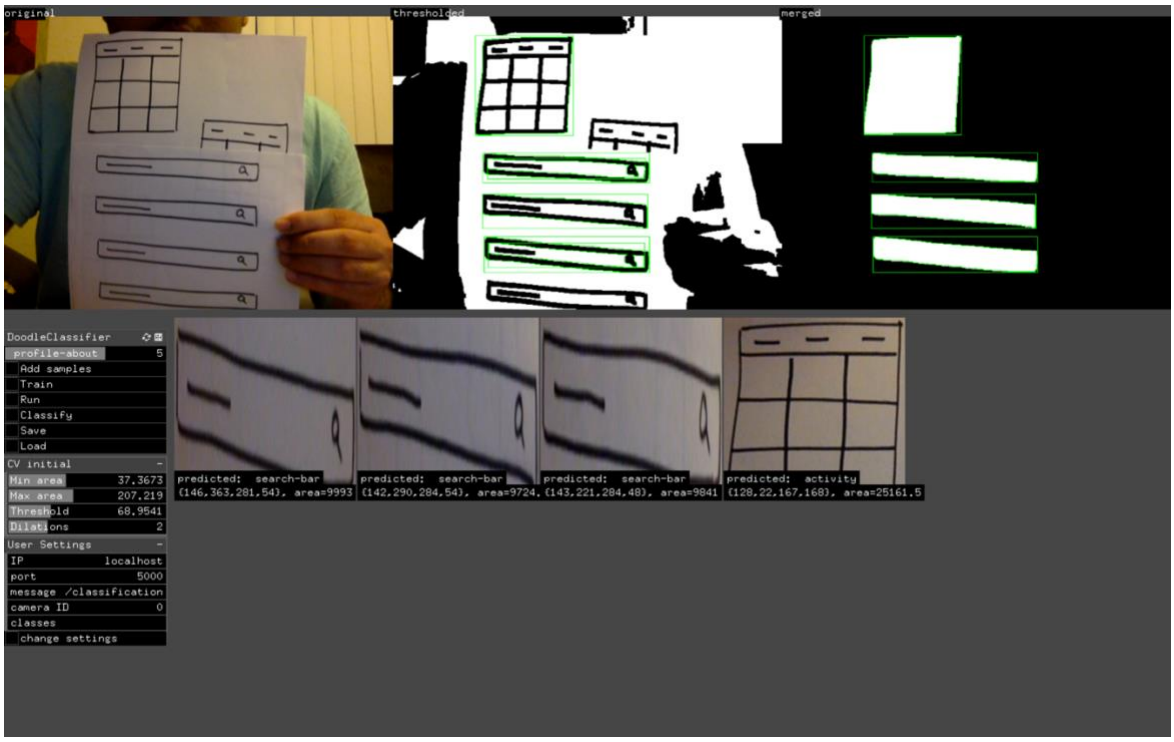


Fig.21 Predicting distinct components

Both components have dashes and rectangles. The search-bar consists of a district search icon. The classifier is able to successfully detect and predict component class and provide meta-data about coordinates about position on bounding box.

Tesseract OCR Experiments:

For these experiments, a variation of tests were conducted. Both global and local properties were tested. Also the size of the handwritten text was varied to measure performance.

Single Component:

Here a single component was selected and only a single instance of properties was written on the designs.

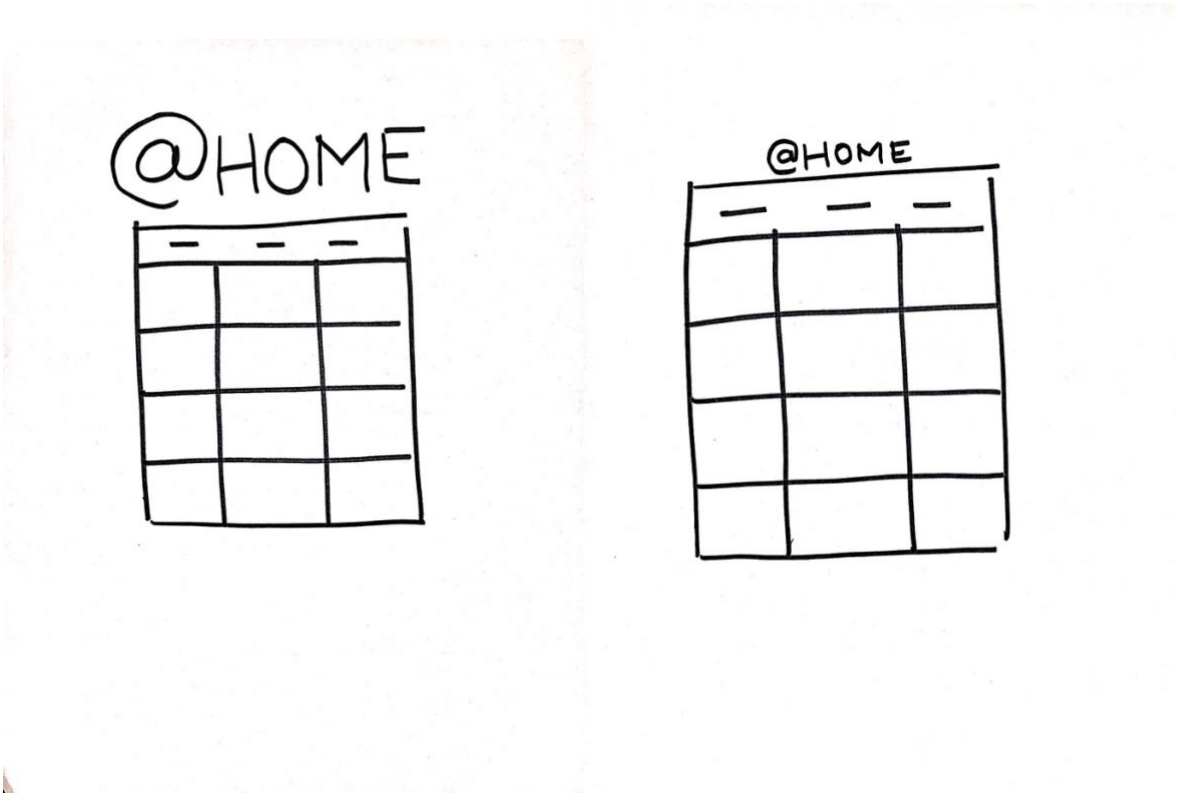


Fig.22 Single Component - Large and Small Text Example

The image-grid component was selected and large text was used to describe component properties. The Tesseract OCR Engine was able to successfully detect and recognize the text written in the large example but failed for the smaller text. Also, examples where the handwritten text deviated from Printed style text, the detections were sometimes incorrect and classified as nearest similar looking character.

Multiple Component:

Here, text is written on more than one component. For the Fig.23, the top-nav and image-card components are used with properties written on the top-right side.

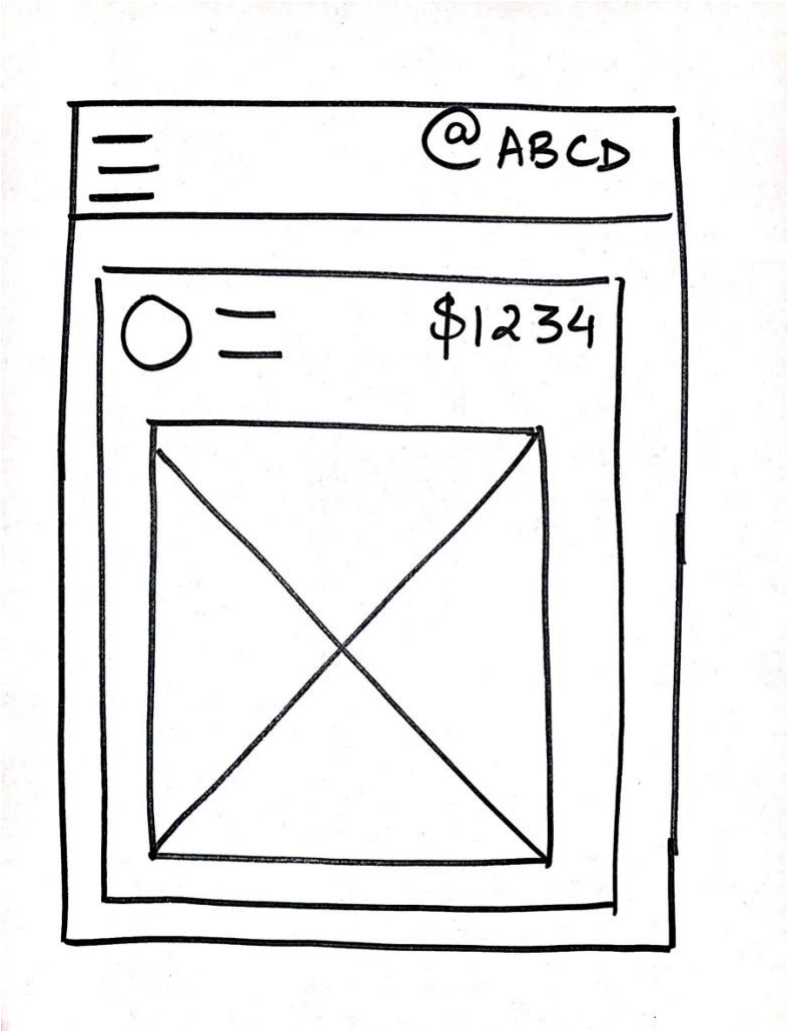


Fig.23 Testing OCR with Multiple Components

The OCR results were poor with complex symbols such as $ and @ unable to be recognized. Also characters spaced close to each other were often misclassified. The output for above Fig.23, were QAEOD and SI334 respectively from top to bottom.

CONCLUSION AND FUTURE WORK

The goal of this project was to develop an end-to-end design assistance tool (YODA) which would help generated code directly from user interface design sketches. Various state-of-the-art object detection and optical character recognition algorithms were researched and finally YOLO along with the Tesseract OCR Engine were used. Also a new design prototyping language compatible with YODA has been presented and implemented.

The UI component classification task powered by YOLO was able to successfully distinguish between components of various size, complexity and shape. The Optical Character Recognition powered by Tesseract OCR Engine showed promising results but is still not mature enough to be integrated into a successful design assistant. Future research for using Tesseract would be applying more image pre-processing and using the deep learning LSTM implementation of the OCR Engine. This way, the OCR can be trained to recognize a user's handwriting instead of trying to compare to printed block letters. The web stack of YODA can be improved to be made more user friendly and usable with the goal of it being used for designers to maintain history of UI designs, facilitate collaboration and add other features similar to popular modern photo-sharing socials applications.

YODA has been demoed to various UI and AI Engineers at Google, Facebook and Airbnb with positive feedback and suggestions. Surprisingly, Airbnb actually have a similar internal tool powered by YOLO suggesting that the direction of YODA and its implementation is near state-of-the-art and has a lot of potential.

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.

[12] Siddharth Kulkarni. (2019, April 28). CS297/298 Research material and blog. Retrieved from http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Fall18/siddharth/

[13] Free Mockup Design Template from https://www.invisionapp.com/inside-design/design-resources/do/

[14] Neuron Image taken with permission from https://www.kdnuggets.com/2016/10/artificial-intelligence-deep-learning-neural-networks-explained.html

[15] Goel, Savita and Savita Dabas. “Vehicle registration plate recognition system using template matching.” *2013 INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING AND COMMUNICATION (ICSC)* (2013): 315-318.

[16] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[17] Jonathan Pedoeem: “YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers”, 2018; [http://arxiv.org/abs/1811.05588 arXiv:1811.05588].