Automatic Generation of Web Page from Hand Drawn Sketches

A project report submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science & Engineering

Submitted by

Yedhin Kizhakkethara
Ranjith R.K

Federal Institute of Science And Technology (FISAT)®
Angamaly, Ernakulam

Affiliated to

APJ Abdul Kalam Technological University
CET Campus, Thiruvananthapuram

FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY (FISAT)

Mookkannoor(P.O), Angamaly-683577

CERTIFICATE

This is to certify that the report entitled "**Automatic Generation of Web Page from Hand Drawn Sketches**" is a bonafide record of the project submitted by **Yedhin Kizhakkethara(FIT16CS125), Ranjith R.K(FIT16CS094)**, in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology (B.Tech) in Computer Science & Engineering during the academic year 2019-20.

Dr. Prasad J C

Staff in Charge | Project Guide | Head of the Department

ABSTRACT

Implementing smooth and engaging experiences for users is a crucial goal for companies of all sizes, and it is a process driven by numerous cycles of prototyping, designing, and user testing. Large organizations and MNCs have the capability and financial power to dedicate entire teams to the design process, which can take several weeks and involve multiple stakeholders. But small startups and businesses do not have these resources, and their user experience may suffer as a result. As webpages are essential for marketing and businesses, there is a need for faster iteration, easier accessibility to non-developers and less dependency on developers for initial prototyping. A deep learning model which can convert hand drawn wireframes of websites into corresponding digital webpages can resolve the mentioned problems.

Contribution by Author

- Yedhin Kizhakkethara
  - Contributed largely to the creation of the Deep Learning model. Wrote the compiler which can convert the GUI tokens into corresponding html code. Guided the creation of the scraper for auto collection of data from the websites. Shared responsibility in creation of the presentations and reports. Wrote test cases and validations, and helped with measuring and mathematically representing the results. Coordinated the overall flow of different technical aspects of the project from start till end.

- Ranjith R.K
  - Collected the huge dataset consisting of nearly 10,000 images and which included a quantifiable proportion of manually drawn sketches too. Performed the preprocessing steps :- cleaning, tagging and labelling the images for further processing. Contributed largely to the creation of the scraper for auto collection of data from the websites. Helped with the theoretical side of creating the Deep Learning model based on the base paper. Helped with the evaluation of the real world application and accuracy of the model. Shared responsibility in creation of the presentations and reports

Yedhin Kizhakkethara
Ranjith R.K

ACKNOWLEDGMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and we are extremely privileged to have got this all along for completion of the project.

We would like to express our deepest appreciation towards Dr. George Issac, Principal, Federal Institute of Science And Technology (FISAT), Dr. Prasad J C, Head of Department, Computer Science and Engineering, for helping us to avail the college facilities all the time.

We owe our deepest gratitude to our project guide Prof. Reshmi R and coordinators Prof. Jestin Joy, Prof. Merin Cherian and Mr. Mahesh C, for their guidance and constant supervision as well as for providing necessary information regarding the project.

We are thankful to all other teaching and non-teaching staff for their support in completion of the project.

Yedhin Kizhakkethara
Ranjith R.K

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

As webpages are essential for marketing and businesses, there's a need for faster iteration, easier accessibility to non-developers and less dependency on developers for initial prototyping. The project aims to create a deep learning model (DNN) which can convert hand drawn sketches created by an user into their corresponding digital webpage, for faster iteration and reduction of dependency on developers for prototyping.

1.1.1 Software Prototyping Model

Prototype Methodology[1] is defined as a Software Development model in which a prototype is built, test, and then reworked when needed until an acceptable prototype is achieved. It also creates a base to produce the final system. The project uses Rapid Throwaway Prototyping strategy, where the prototyping is quickly developed to show how the requirement will look visually. The customer's feedback helps drives changes to the requirement, and the prototype is again created until the requirement is satisfied.

1.1.2 Website Prototyping

Website prototypes[2] are demos of the websites early in the project life cycle. They play an important role in the project life cycle. These prototypes are generally used to have an idea about what the the clients want and to show them how their final output will be structured. In many occasions the website developer and designer of the website may not be a single person. Websites are usually developed by web developers, who may not be very good with designing and aesthetics. So typically a prototype will be made by designers, who do not have a good idea about web development, in order to convey what exactly a client wants. Then web developers will work on the prototype done by the designers. This is a very time consuming process.

1.1.3 Deep Neural Network

A deep neural network (DNN)[3] represents the type of machine learning where the system uses many layers of nodes to derive higher-level functions from input information. It means transforming the data into a more creative and abstract component. The DNN finds the correct mathematical manipulation to turn the

input into the output, whether it be a linear relationship or a non-linear relationship. The network moves through the layers calculating the probability of each output. For example, a DNN that is trained to recognize dog breeds will go over the given image and calculate the probability that the dog in the image is a certain breed. The user can review the results and select which probabilities the network should display (above a certain threshold, etc.) and return the proposed label. Each mathematical manipulation as such is considered a layer, and complex DNN have many layers, hence the name "deep" networks.

1.2 Problem Statement

Creating smooth and engaging web experiences for users is a critical goal for companies of all sizes. The usual design workflow is as follows:-

- Product managers develop the requirements
- Designers use those requirements and create variations of mockups and adds the changes manually based on feedback
- Developers implement those designs into code and ship the product to end users

Since the duration of the development cycle and the phase of creating prototypes for quick iterations are very time consuming, there is a need for a better system in place, which can help with faster iteration and less dependency on developers. A deep learning model which can convert hand drawn wireframes of websites into corresponding digital webpages can resolve the mentioned problems.

1.3 Objective

- Automatic transformation of hand drawn sketches created by a designer to a web page.
- Reduce the dependency on actual web developers for initial prototyping of a webpage.
- Enable fast and easier web page prototype development for non-developers and businesses.
- Use of deep learning techniques for realtime conversion of sketches to web page for prototyping.
- Provide an intuitive User Interface and experience for the designer to customize the web page prototype by making each of the elements in the webpage manipulatable by barebones html/css alone.
- When machine learning is used to power front-end development, the designers and developers get more time to focus on creative tasks. Eventually, the optimization of time, resources and design budgets **can help companies to reduce costs and increase the return on investment (ROI).**

Chapter 2

Related works

- Sketch2Code
  - Sketch2Code[4] discusses the usage of two methods, a classical computer vision technique and a deep segmentation method.
  - The paper presents approaches which automates the process of generation of website from sketch.
  - The classical computer vision technique follows several steps which include Element Detection, Structural Detection, Container Classification and Layout Normalization. This method's image classifier was very highly discriminatory.
  - Generally the DNN based method outperform the results of the classical computer vision model with the exception of the paragraph class.
  - The precision score of the paragraph element performed worse in the deep learning model compared with the classical computer vision model (0.562 vs. 0.747). This indicates that the classical computer vision model's paragraph detection produced fewer false positives.
- Pix2Code
  - Pix2Code[5] discusses about a novel approach based on Convolutional and Recurrent Neural Networks allowing the generation of computer tokens from a single GUI screenshot as input.
  - That is, no engineered feature extraction pipeline nor expert heuristics was designed to process the input data: the model learns from the pixel values of the input image alone.
  - Deep Learning methods such as CNN and LSTM were leveraged to train a model end-to-end so as to automatically generate code from screenshots of code.
- Canny Text Detector: Fast and Robust Scene Text Localization Algorithm
  - Canny Text Detector: Fast and Robust Scene Text Localization Algorithm[6], presents a text detection algorithm, Canny Text Detector, which takes advantage of the similarity between image edge and text for effective text localization with improved recall rate.
  - The algorithm makes use of double threshold and hysteresis tracking to detect texts of low confidence.

  - As closely related edge pixels construct the structural information of an object, the paper observes that cohesive characters compose a meaningful word/sentence sharing similar properties such as spatial location, size, color, and stroke width regardless of language.
  - Experimental results on public datasets demonstrate that the algorithm outperforms some of the state of the art scene text detection methods in terms of detection rate.

- Reverse Engineering Mobile Application User Interfaces with REMAUI
  - When developing the user interface code of a mobile application, in practice a big gap exists between the digital conceptual drawings of graphic artists and working user interface code.
  - Currently, programmers bridge this gap manually, by which can be cumbersome and expensive.
  - To bridge this gap, REMAUI[7] can be used. On a given input bitmap REMAUI identifies user interface elements such as images, texts, containers, and lists, via computer vision and optical character recognition (OCR) techniques.
  - In the experiments conducted on 488 screenshots of over 100 popular third-party Android and iOS applications, REMAUI-generated user interfaces were similar to the originals, both pixel-by-pixel and in terms of their runtime user interface hierarchies. REMAUI's average overall runtime on a standard desktop computer was 9 seconds.

Chapter 3

Design

The proposed work deals with creation of a model that would be able to take a simple hand-drawn prototype of a website design, and instantly generate a working HTML website from that image. The design of the model was heavily based on the model from the base paper, pix2code. The design leveraged very popular and heavily developed machine learning techniques and algorithms, and was divided into several stages for overall completion of the model. At the core a classic CNN based computer vision model extracts image features from the source images. Then a language model consisting of a Gated Recurrent Unit (GRU)[8] encodes sequences of source code tokens. Then a decoder model (also a GRU), takes in the output from the previous two steps as its input, and predicts the next token in the sequence.

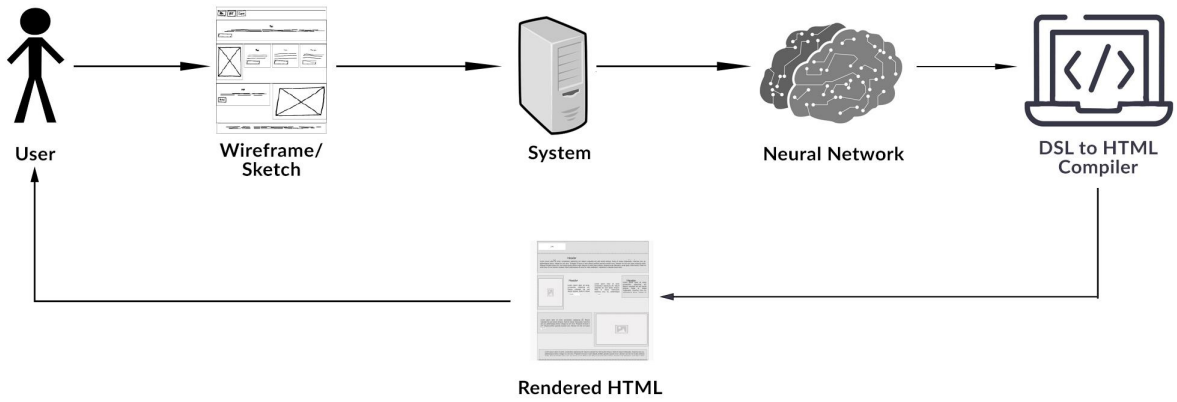

Figure 3.1: UML Diagram

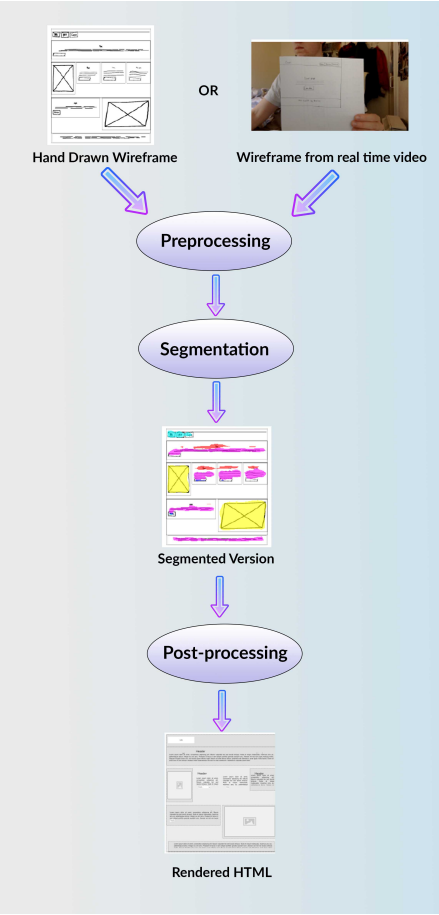

Figure 3.2: High Level Overview

3.1 Image Captioning

The problem the model aimed to solve fell under a broader umbrella of tasks known as program synthesis, which is the automated generation of working source code. Though much of program synthesis deals with code generated from natural language specifications or execution traces, in current projects context the model could leverage the fact that a source image (the hand-drawn wireframe) was available to start with. Image captioning[9] is a well-studied domain in machine learning which seeks to learn models that tie together images and text, specifically to generate descriptions of the contents of a source image. This idea was one of the core ideas that the project used, since the drawn website wireframe can be considered as the input image and its corresponding HTML code as its output text.

3.2 Classic CNN

Convolutional Neural Networks[10] are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class

scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

Contents of a classic Convolutional Neural Network:-

- Convolutional Layer.
- Activation operation following each convolutional layer.
- Pooling layer especially Max Pooling layer and also others based on the requirement.
- Finally Fully Connected Layer.

3.3 Procedures for Training and Sampling

3.3.1 Training

Algorithm 1: Training the Model

Input: Training Dataset consisiting of nearly 10000 processed images
Output: A CNN based model

```
/* Initalize the model                                          */
Load the dataset which includes processed images and vocabulary
Initialize a Image Encoder using Sequential Conv2D network
Initialize a Language Encoder by embedding the vocabulary and adding
  two GRUs
Initialize a Decoder by concatenating the image and language models,
  with a softmax layer
Compile the model
/* Extract the features                                         */
while Dataset is not empty do
    extract image features and text features
    if data is image then
        create an array format of the image sequence
        add sequence as a new key in the feature map
    if data is text then
        prepend <START> and append <END> to the sequence
        add the sequence as the value in feature map
    append the feature map to an feature array
Fit the model with the feature maps and start training
Save the model in json format once completed, for easier reusability
```

3.3.2 Sampling

Algorithm 2: Sampling on a test image using the trained Model

Input: HandDrawn sketch
Output: Predicted GUI code and compiled HTML code

```
Read the input image
Load the already trained model
/* Generating GUI                                                    */
Extract image features using opencv
in_text := '<START> '
while i < MAX_SEQ do
    Tokenize in_text to sequences
    Pad sequences to same length
    Predict the token using the trained model with sequences and image
     features
    if generated token is not NULL then
        Append it to in_text
    else
        if generated token is '<END>' then
            break
generated_gui := split in_text using whitespace as delimiter
Save it as the output gui file
/* Generating HTML from GUI                                          */
Open the saved GUI prediction and compile it into HTML
if compilation successfull then
    Save it as a html file
    Display the webpage
```

3.4 System Requirements

3.4.1 Hardware Requirements

1. GPU for training with large dataset

3.4.2 Software Requirements

1. Model Creation Modules
   (a) opencv
   (b) keras
   (c) numpy
   (d) tensorflow
   (e) h5py
2. Frontend

   (a) HTML
   (b) CSS

3. Dataset Generation
   (a) opencv
   (b) pillow
   (c) scrapy

4. Evaluation and Testing
   (a) nltk
   (b) pytest

3.5 Implementation

The implementation involved several steps which were divided and finished off one by one in order.

3.5.1 Dataset Collection and Preprocessing

Refer appendix B for code details.

The aim was to create a diverse collection of hand drawn wireframe sketches and their corresponding HTML code. For training the model to accurately figure out the components from a given image, a higher amount of dataset was required. The ideal dataset is the one with thousands of hand drawn wireframe sketches from a large number of people for attaining a diverse collection. But due to the time constraint, it was not practical to manually draw thousands of wireframe sketches on paper and create their HTML code equivalents. So different methods were used for creating the dataset.

- The already existing dataset of research paper 'pix2code' was used as starting point. It was an open-source dataset from the pix2code paper, which consisted of 1,750 screenshots of synthetically generated websites and their relevant source code. Each image of the generated website in the dataset consisted of combinations of just a few simple html element and sections such as buttons, headers, and divs. Each computer generated screenshot is paired with its corresponding GUI code. GUI code consisted of tokens corresponding to the screenshots. This GUI code can be later compiled to HTML or any other domain specific language. This dataset needed was modified according to the needs of this project, that is the computer generated screenshots were modified to make it look like they were drawn by hand. Open CV and PIL libraries were used for this task. The straight line borders of the elements were converted to curves, thickness of the borders were increased and the font was changed in order to make it look handwritten.

- The dataset needed to have a more accurate representation of the real world. So real hand drawn images needed to be there in the dataset. Thus the

elements of the website like headers, buttons etc were hand drawn separately and all of these were combined by leveraging OpenCV and PIL libraries in python, to create a augmented hand drawn wireframe image and replicate these images by adding skews, shifts, and rotations to mimic the variability in actual drawn sketches.

- The final method was stylesheet modification of hand picked websites. The following changes were mostly applied:-
  - Changed the border radius of elements on the page to curve the corners of buttons and divs.
  - Adjusted the thickness of borders to mimic drawn sketches, and added drop shadows.
  - Changed the font to one that looks like handwriting.

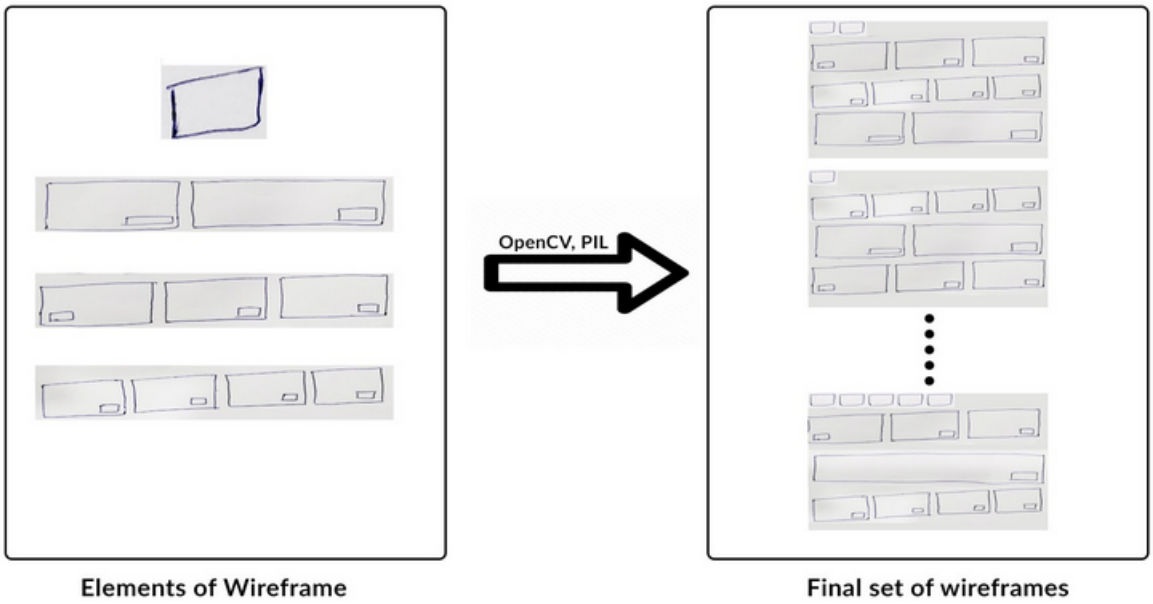

Figure 3.3: Dataset Generation using augmentation

3.5.2 Model Creation

Refer appendix A for code details.

The entire machine learning algorithms were leveraged from Keras library. The model was created based on a classic CNN which is primarily used in image captioning techniques. The classic CNN model is a sequential model which stacks layer by layer. We stacked the model with 6 convolutional layers, and performed RELU[11] activation on each of the layer. RELU was chosen in order to have achieve faster training, more sparsity and a reduced likelihood of vanishing gradient. For summarizing the features, we used a max pooling layer, with a pool size of 2 and 2 strides. This model acted as the vision model in architecture.

Language Model Creation

In order to hot encode the GUI context and associate it with the features extracted using the vision model, a language model consisting of a Gated Recurrent Unit (GRU) that encodes sequences of source code tokens was created.

Addition of Embedding Layer

For improved accuracy of the model, custom word embeddings were also used. Instead of using one-hot vectors to represent the words, the low-dimensional vectors learned using custom vocabulary carry semantic meaning similar words have similar vectors. Using these vectors is a form of pre-training. Keras library directly provides a class for Embedding, which was leveraged for the same.

Addition of second GRU layer

In order to allow the model to capture higher-level interactions, a second layer of GRU was added to the network. The addition of GRU was primarily to help with the vanishing gradient problem, since the dataset was large in size.

Decoder

The decoder model was also a GRU, which took in the output from the previous two steps as its input, and predicted the next token in the sequence.

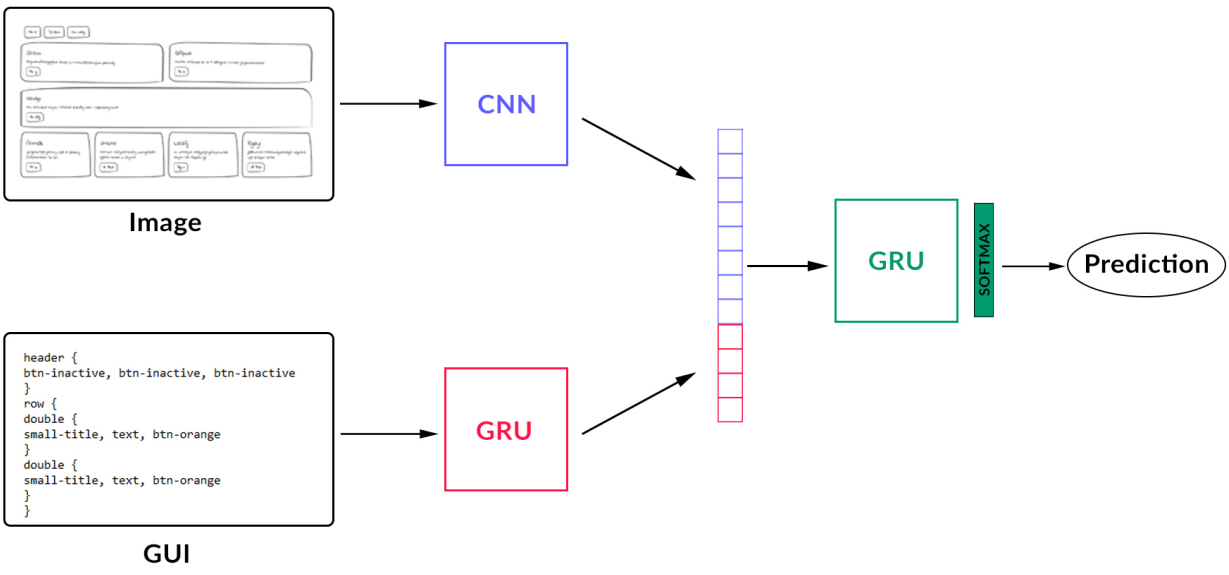

Figure 3.4: Model Architecture

3.5.3 Training and Sampling

To train the model, the source code was split into sequences of tokens. A single input for the model is one of these sequences along with its source image, and its label is the next token in the document. The model used the cross-entropy cost as its loss function, which compares the model's next token prediction with the actual next token. At inference time when the model is tasked with generating code from scratch, the process is slightly different. The image is still processed through the CNN network, but the text process is seeded with just a starting sequence. At each step, the model's prediction for the next token in the sequence is appended to the current input sequence, and fed into the model as a new input sequence. This is repeated until the model predicts an `<END>` token or the process reaches a predefined limit to the number of tokens per document. Once the set of predicted tokens was generated from the model, a custom compiler converted the DSL tokens into HTML, which can be rendered in any browser. Therefore during sampling phase, the user only needs to provide an image of the hand drawn sketch to the model, from whence the image gets preprocessed through some filters and then the model can generate the GUI context corresponding to the image and then the custom compiler can use it to generate code in any Domain Specific Language(DSL)[12].

Chapter 4

Results

The output of the project is a deep learning model which can generate html code from a hand drawn wireframe of the website. Upon conducting a test with nearly 20 hand drawn images, in which 50% were part of the training and rest were custom out of training hand drawn sketches, the model was able to accurately generate a accurate website for about 75% of the test images.

4.1 BLEU Score

BLEU (bilingual evaluation understudy)[13] is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. Quality is considered to be the correspondence between a machine's output and that of a human: "the closer a machine translation is to a professional human translation, the better it is"; this is the central idea behind BLEU. Scores are calculated for individual translated sentences by comparing them with a set of good quality reference translations. Those scores are then averaged over the whole corpus to reach an estimate of the translation's overall quality. BLEU's output is always a number between 0 and 1. This value indicates how similar the candidate text is to the reference texts, with values closer to 1 representing more similar texts.

4.2 Evaluation

The model was evaluated using the BLEU score. When providing the image, the user will know exactly what should be the generated DSL token or the GUI context corresponding to that image. Once the image is provided to the model, it will predict and output a GUI context based on the features extracted from the image. So comparing the similarity by using BLEU score between the GUI context provided by user and the one dynamically generated by the model can provide a score which is a accurate evaluation of the quality of the model.

Table 4.1: Result of evaluation of Model using test images

Image Type	BLEU Score	Percentage
From Training Set	0.9744	97.44
Outside Training Set	0.7102	71.02

Figure 4.1: BLEU Score(0.9744) : When image provided was from training set itself

Figure 4.2: BLEU Score(0.7102) : When image with high level of inconsistency was provided from outside training set

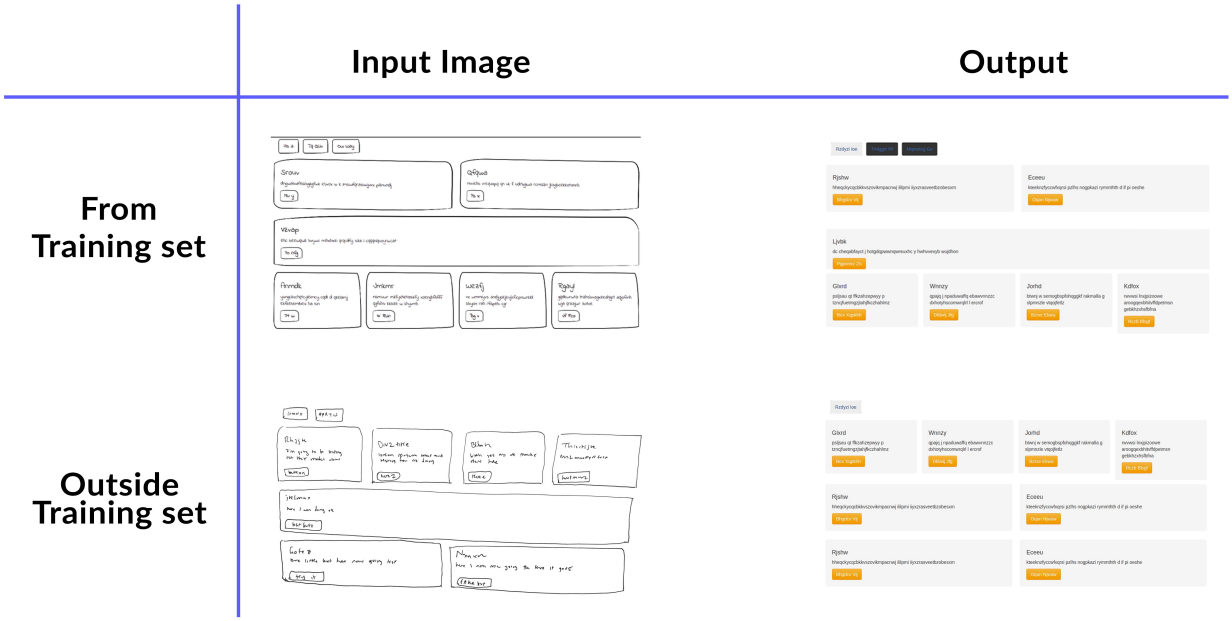

Figure 4.3: Input-Output Comparison with varying test cases : Refer appendix C

4.3 Comparison With Existing Systems

The Method using classical computer vision technique shows very high discrimination and another highlighted result is that it shows many false positives based on the data. This is due to container elements being misclassified as input elements, indicating the need for better post detection filtering.

The method using DNN generally outperforms the results of the classical computer vision method. This shows that the DNN method is better efficient, less discriminatory and have lesser chance for false positives. The significantly higher F1 score[14] on classification of all elements except paragraphs indicates that the implementation of deep learning segmentation outperforms classical computer vision techniques at the task of element detection and classification.

On the other hand the paper pix2code only works for screenshot images, where each element position and structure is well defined, with corresponding element description within the bounding boxes. This is the major disadvantage pix2code has when comparing to sketch2code.

Reverse Engineer Mobile Application User Interfaces (REMAUI) is a technique which can be used to bridge the gap between digital conceptual drawings of graphic artists and mobile user interface code. But the model's precision suffered if a subject contained a bitmap that contained both text and non-text imagery because OCR it was hard to distinguish if a given text is plain text or belongs to

a bitmap of text and other elements. Another disadvantage was that low image recall occurred when images overlapped and were of similar colour or the top image was somewhat transparent. These scenarios are challenging for edge detection.

Canny text detector presents a novel scene text detection algorithm which takes advantage of the similarity between image edge and text for effective text localization with improved recall rate. This method can be used for text detection, which can be used in the proposed model for various purposes including the detection of writings from the user input.

Chapter 5

Conclusion

Creating intuitive and engaging experiences through website for users is a critical goal for companies of all sizes, and it's a process driven by quick cycles of prototyping, designing, and user testing. The main aim of the project was to use modern deep learning algorithms to create a model which can significantly streamline the design workflow and empower any business to quickly create and test webpages. This project successfully created a CNN based model, which can generate html code corresponding to the hand drawn sketch of a website provided by the user.

5.1 Relevance of Project

5.1.1 Advantages of Website Prototyping

- To gain agreement on what is in and out of scope
- To generate support or even investment for the project
- To test theories and ideas regarding layout and structure of the website
- More importantly to gather user feedback through usability testing

5.1.2 Relevance of an Automated Model

- Reduce the dependency on actual web developers for initial prototyping[1] of a webpage.
- Enable fast and easier web page prototype development for non-developers and businesses.
- When machine learning is used to power front-end development, the designers and developers get more time to focus on creative tasks. Eventually, the optimization of time, resources and design budgets **can help companies to reduce costs and increase the return on investment (ROI).**

5.2 Future Work

Although the model was able to generate html code from the given hand drawn sketch input, the accuracy and flexibility of output based on varied inputs had several limitations, which could be improved in future, like:

- The model can't classify all elements of the website since the model was trained on a set of pre-defined html element vocabulary. Therefore providing more website examples which includes elements like images, dropdown menus, forms etc would be an enhancement.
- The dataset count was pretty low considering the fact that the hand drawn sketch provided as input could be heavily inconsistent when a user draws it. Therefore scraping more real world websites and generating corresponding hand drawn training sample is necessary.
- Also another way to generate more variations in the hand-drawn sketch data for training would be to leverage a Generative Adversarial Network(GAN)[15], so that the model could be much more accurate on inconsistent inputs.

Bibliography

[1] Smith MF Software Prototyping: Adoption, Practice and Management. McGraw-Hill, London (1991).

[2] Neil Young Website Prototyping : https://www.experienceux.co.uk/faqs/what-is-a-website-prototype/

[3] Schmidhuber J(2015). "Deep Learning in Neural Networks: An Overview". Neural Networks. 61: 85117. arXiv:1404.7828. doi:10.1016/j.neunet.2014.09.003. PMID 25462637. S2CID 11715509.

[4] Alex Robinson. "Sketch2code: Generating a website from a paper mockup". In: CoRR abs/1905.13750 (2017). arXiv: 1905.13750. url: https://arxiv.org/abs/1905.13750

[5] Tony Beltramelli. 2018. pix2code: Generating Code from a Graphical User Interface Screenshot. In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '18). ACM, New York, NY, USA, Article 3, 6 pages. DOI: https://doi.org/10.1145/3220134.3220135

[6] H. Cho, M. Sung, and B. Jun, Canny text detector: Fast and robust scene text localization algorithm," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Mar. 2016, pp. 35663573.

[7] Tuan A. Nguyen and Christoph Csallner. 2015. Reverse engineering mobile application user interfaces with REMAUI. In Proc. 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 248259.

[8] "Recurrent Neural Network Tutorial, Part 4 Implementing a GRU/LSTM RNN with Python and Theano WildML". Wildml.com. 2015-10-27. Retrieved May 18, 2016.

[9] Pranoy Radhakrishnan Image Captioning in Deep Learning : https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2

[10] "Convolutional Neural Networks (LeNet) DeepLearning 0.1 documentation". DeepLearning 0.1. LISA Lab. Retrieved 31 August 2013.

[11] Romanuke, Vadim (2017). "Appropriate number and allocation of ReLUs in convolutional neural networks". Research Bulletin of NTUU "Kyiv Polytechnic Institute". 1: 6978. doi:10.20535/1810-0546.2017.1.88156.

[12] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. ACM Computing Surveys, 37(4):316344, 2005.doi:10.1145/1118890.1118892

[13] Papineni, K.; Roukos, S.; Ward, T.; Zhu, W. J. (2002). BLEU: a method for automatic evaluation of machine translation (PDF). ACL-2002: 40th Annual meeting of the Association for Computational Linguistics. pp. 311318. CiteSeerX 10.1.1.19.9416.

[14] Derczynski, L. (2016). Complementarity, F-score, and NLP Evaluation. Proceedings of the International Conference on Language Resources and Evaluation.

[15] Goodfellow, Ian; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua (2014). Generative Adversarial Networks. Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 26722680.

Appendices

Appendix A

Model Training

```
def train_model(self):
    self.vocab_size = data.vocabulary()[1]

    # Create Image Encoder
    image_caption_model = Sequential()
    image_caption_model.add(Conv2D(16, (3, 3),
        padding='valid', activation='relu',
        input_shape=(256, 256, 3,)))
    image_caption_model.add(Conv2D(16, (3,3),
        activation='relu',
        padding='same', strides=2))
    image_caption_model.add(Conv2D(32, (3,3),
        activation='relu',
        padding='same'))
    image_caption_model.add(Conv2D(32, (3,3),
        activation='relu',
        padding='same', strides=2))
    image_caption_model.add(Conv2D(64, (3,3),
        activation='relu',
        padding='same'))
    image_caption_model.add(Conv2D(64, (3,3),
        activation='relu',
        padding='same', strides=2))
    image_caption_model.add(Conv2D(128, (3,3),
        activation='relu',
        padding='same'))
    image_caption_model.add(Flatten())
    image_caption_model.add(Dense(1024, activation='relu'))
    image_caption_model.add(Dropout(0.3))
    image_caption_model.add(Dense(1024, activation='relu'))
    image_caption_model.add(Dropout(0.3))
    image_caption_model.add(RepeatVector(MAX_LENGTH))
```

```
    input_img = Input(shape=(256, 256, 3,))
    encoded_image = image_caption_model(input_img)

    # Create the Language encoder
    language_input = Input(shape=(MAX_LENGTH,))
    language_model = Embedding(vocab_size, 50,
                    input_length=MAX_LENGTH,
                    mask_zero=True)(language_input)
    language_model = GRU(128,
                    return_sequences=True)(language_model)
    language_model = GRU(128,
                    return_sequences=True)(language_model)

    # Create the Decoder
    decoder = concatenate([encoded_image, language_model])
    decoder = GRU(512, return_sequences=True)(decoder)
    decoder = GRU(512, return_sequences=False)(decoder)
    decoder = Dense(vocab_size, activation='softmax')(decoder)

    # Compile the overall model
    self.model = Model(inputs=[visual_input,
                language_input], outputs=decoder)
    optimizer = RMSprop(lr=0.0001, clipvalue=1.0)
    self.model.compile(loss='categorical_crossentropy',
                optimizer=optimizer, metrics=['accuracy'])

```

Appendix B

Custom dataset generation from scratch using augmentation

```
class GenDataset:
    def __init__(self, ASSETS_PATH, OUTPUT_PATH):
        self.background = Image.open(
            ASSETS_PATH + "white_background.png"
        )
        self.gui_path = "../all_data/data/*.gui"
        self.row_count = 0

        self.header_btn = (
            ASSETS_PATH + "headers/header_button"
        )
        self.row1_single = (
            ASSETS_PATH
            + "row1/first_row_single.png"
        )
        self.row1_double = (
            ASSETS_PATH
            + "row1/first_row_double.png"
        )
        self.row1_triple = (
            ASSETS_PATH
            + "row1/first_row_triple.png"
        )
        self.row1_quad = (
            ASSETS_PATH
            + "row1/first_row_quad.png"
        )

        self.row2_single = (
            ASSETS_PATH
```

```
            + "row2/second_row_single.png"
        )
        self.row2_double = (
            ASSETS_PATH
            + "row2/second_row_double.png"
        )
        self.row2_triple = (
            ASSETS_PATH
            + "row2/second_row_triple.png"
        )
        self.row2_quad = (
            ASSETS_PATH
            + "row2/second_row_quad.png"
        )

        self.row3_single = (
            ASSETS_PATH
            + "row3/third_row_single.png"
        )
        self.row3_double = (
            ASSETS_PATH
            + "row3/third_row_double.png"
        )
        self.row3_triple = (
            ASSETS_PATH
            + "row3/third_row_triple.png"
        )
        self.row3_quad = (
            ASSETS_PATH
            + "row3/third_row_quad.png"
        )

    def traverseFiles(self):
        count = 0
        for filename in tqdm(glob(self.gui_path)):
            self.row_count = 0
            self.background = Image.open(
                ASSETS_PATH
                + "white_background.png"
            )
            print(
```

```
                f"""
{filename} -> {OUTPUT_PATH}{count}.png
            """
            )
            count += 1
            with open(filename) as file:
                itr = 0
                lines = file.read().splitlines()
                length = len(lines)
                while itr < length:
                    if "header" in lines[itr]:
                        itr += (
                            1
                        )  # move to line with btn-inactive
                        btn_count = len(
                            re.findall(
                                "btn-inactive",
                                lines[itr],
                            )
                        )
                        self.overlay(
                            btn_count,
                            f"{self.header_btn}_{btn_count}.png",
                        )
                        itr += (
                            2
                        )  # move to other containers after header
                    elif re.search(
                        r"row", lines[itr]
                    ):
                        itr += 1
                        self.row_count += 1
                        if "double" in lines[itr]:
                            path_variable = getattr(
                                dataset_generator,
                                f"row{self.row_count}_double",
                            )
                            self.overlay(
                                btn_count,
                                path_variable,
                            )
```

```
                            itr += 6
                        elif (
                            "single" in lines[itr]
                        ):
                            path_variable = getattr(
                                dataset_generator,
                                f"row{self.row_count}_single",
                            )
                            self.overlay(
                                btn_count,
                                path_variable,
                            )
                            itr += 4
                        elif (
                            "triple" in lines[itr]
                        ):
                            path_variable = getattr(
                                dataset_generator,
                                f"row{self.row_count}_triple",
                            )
                            self.overlay(
                                btn_count,
                                path_variable,
                            )
                            itr += 10
                        elif (
                            "quadruple"
                            in lines[itr]
                        ):
                            path_variable = getattr(
                                dataset_generator,
                                f"row{self.row_count}_quad",
                            )
                            self.overlay(
                                btn_count,
                                path_variable,
                            )
                            itr += 13
                    else:
                        itr += 1
                    if itr + 4 > length:
```

```
                        break
                    print(
                        f"""
{itr} < {length} : {lines[itr]}
                    """
                    )
            self.background.show()
            if count == 1:
                break

    def overlay(self, times, fg_path):
        """
        foreground can be either of these:-
        1) header buttons - btn-inactive
        2) row1 - single, double, triple, quadruple
        3) row2 - single, double, triple, quadruple(optional)
        4) row3 - single, double, triple, quadruple(optional)

        First parameter to .paste() is the image to paste.
        Second are coordinates, and the secret
        sauce is the third parameter.
        It indicates a mask that will be used
        to paste the image.
        If you pass a image with transparency,
        then the alpha channel is used as mask.
        """
        foreground = Image.open(fg_path)
        self.background.paste(
            foreground, (0, 0), foreground
        )

```

Appendix C

BLEU Score calculation using nltk library

```
class Evaluator:
    def __init__(self):
        pass

    @classmethod
    def get_sentence_bleu(cls, original_gui_filepath,
    generated_gui_filepath):
        original_gui = Evaluator.load_gui_doc(
    original_gui_filepath)
        generated_gui = Evaluator.load_gui_doc(
    generated_gui_filepath)
        hypothesis = generated_gui[1:-1]
        reference = original_gui
        references = [reference]
        return sentence_bleu(references, hypothesis)

    @classmethod
    def load_gui_doc(cls, gui_filepath):
        file = open(gui_filepath, 'r')
        gui = file.read()
        file.close()
        gui = ' '.join(gui.split())
        gui = gui.replace(',', ' ,')
        gui = gui.split()
        btns_to_replace = ['btn-green', 'btn-red']
        normalized_gui = ['btn-orange' if token in
            btns_to_replace else token for token in gui]
        normalized_gui = ['btn-active' if token == 'btn-inactive'
    else token for token in normalized_gui]
        return normalized_gui

```