Automatic Generation of Web Page from Hand Drawn Sketches

by

Yedhin Kizhakkethara(FIT16CS125)

Ranjith R.K(FIT16CS094)

June 16, 2020

Abstract

- 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 help in achieving this goal.

Problem Statement

- 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.

Problem Statement

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

Problem Statement

- 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.

Introduction

- Website prototyping[3] is part of almost every business/consultancy and software development industries.
- Essentially, a website prototype allows the project stakeholders to see what the final product will look like early in the project lifecycle.

Expected Outcome

- Rapid generation of an accurate design of the webpage (in html form) from the rough hand drawn sketches.

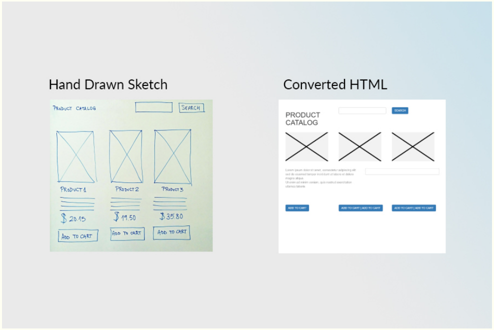


Figure 1: Expected Outcome

Architecture

UML Diagram

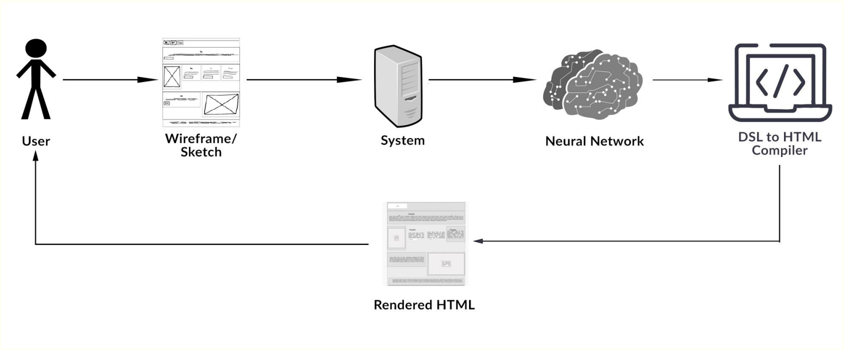


Figure 2: UML Diagram

High Level Overview

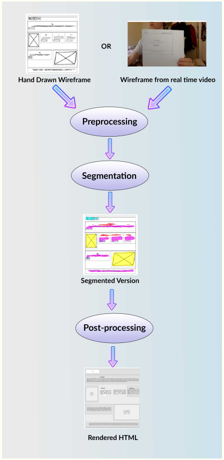
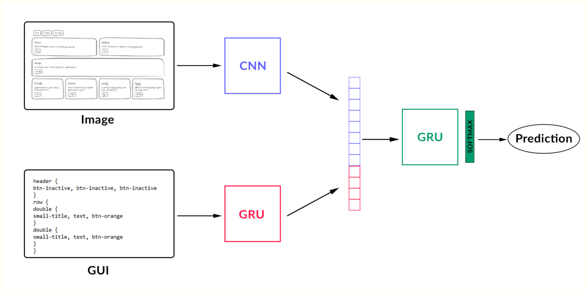


Figure 3: Overview of the workflow

System Architecture

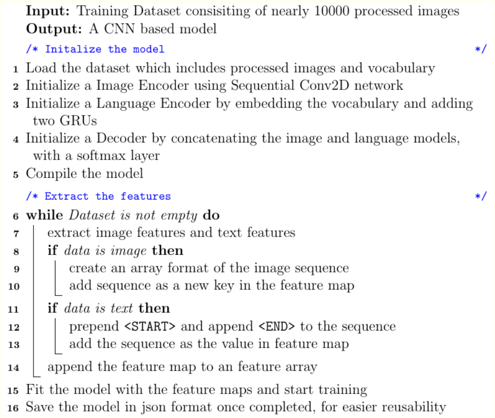


```
header {
btn-inactive, btn-inactive, btn-inactive
}
row {
double {
small-title, text, btn-orange
}
double {
small-title, text, btn-orange
}
}
```

Figure 4: Model Architecture

Algorithms

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
```

Sampling the Model

```
   Input: HandDrawn sketch
   Output: Predicted GUI code and compiled HTML code
1  Read the input image
2  Load the already trained model
   /* Generating GUI                                                    */
3  Extract image features using opencv
4  $in\_text$ := '<START> '
5  while $i < MAX\_SEQ$ do
6      Tokenize $in\_text$ to sequences
7      Pad sequences to same length
8      Predict the token using the trained model with sequences and image
        features
9      if generated token is not NULL then
10         Append it to $in\_text$
11     else
12         if generated token is '<END>' then
13             break
14 $generated\_gui$ := split $in\_text$ using whitespace as delimiter
15 Save it as the output gui file
   /* Generating HTML from GUI                                          */
16 Open the saved GUI prediction and compile it into HTML
17 if compilation successfull then
18     Save it as a html file
19     Display the webpage
```

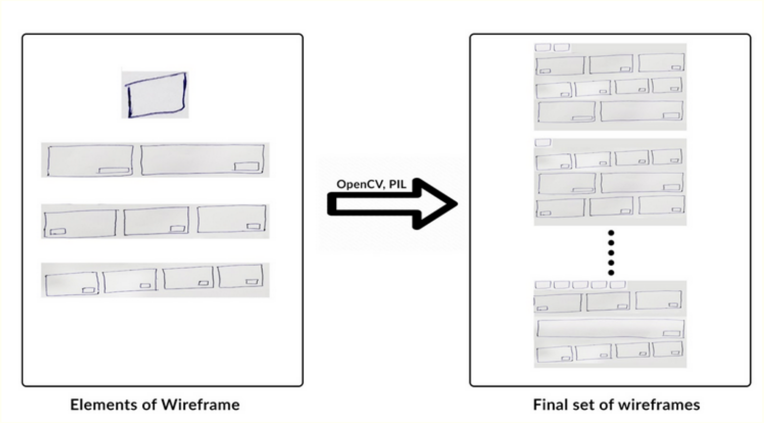
Implementation

Dataset Generation

Several methods were used to create a diverse collection of hand drawn wireframe sketches and their corresponding GUI code for training.

- Used the open-source dataset from the pix2code[5] paper, which consisted of 1,750 screenshots of synthetically generated websites and their relevant source code.
- In order to create more custom hand drawn sketches, elements of the website like headers, buttons etc were hand drawn separately and all of them 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.

Dataset Generation II



Model Creation

The machine learning algorithms used in the project were leveraged from the Keras library. The model creation included several parts, like:-

- Vision Model Creation
- Language Model Creation
- Decoder

Vision Model

- The model was created based on a classic CNN[9] which is primarily used in image captioning techniques.
- The model was stacked with 6 convolutional layers, as done in the base paper, and performed RELU[1] 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, a max pooling layer was used, with a pool size of 2 and 2 strides.

Language Model

- A language model consisting of a Gated Recurrent Unit (GRU)[8] that encoded GUI context and associated it with features extracted from the vision model was created.
- For improved accuracy of the model, custom word embeddings were also used.
- In order to allow the model to capture higher-level interactions, a second layer of GRU was added to the network.

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.
- A softmax layer then provides the likelyhood of each class in this multi-class problem.

Results

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.
- The model was evaluated using the BLEU[11] score.

Table 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

Results II

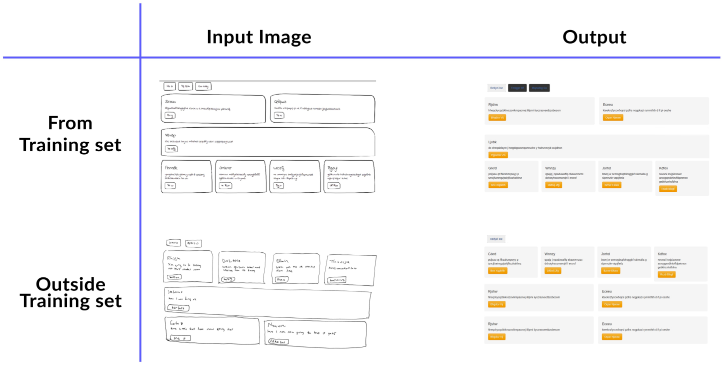


Figure 5: BLEU Scores : 0.9744 for training set image and 0.7102 for external set

Comparison With Existing System

Comparison With Existing System

- 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.

Comparison With Existing System

- 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[2] 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.

Comparison With Existing System

- On the other hand the paper pix2code[5] 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.

Comparison With Existing System

- Reverse Engineer Mobile Application User Interfaces (REMAUI)[7] 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.

Comparison With Existing System

- Canny text detector paper[6] 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.

Conclusion

Relevance of the Project

- 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.
- 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).**

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.

Future Work

- 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)[12], so that the model could be much more accurate on inconsistent inputs.

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.

References

- Romanuke, Vadim (2017). ”Appropriate number and allocation of ReLUs in convolutional neural networks”. Research Bulletin of NTUU ”Kyiv Polytechnic Institute”. 1: 69–78. doi:10.20535/1810-0546.2017.1.88156.
- Derczynski, L. (2016). Complementarity, F-score, and NLP Evaluation. Proceedings of the International Conference on Language Resources and Evaluation.
- Neil Young Website Prototyping : https://www.experienceux.co.uk/faqs/what-is-a-website-prototype/
- 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

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

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. 3566–3573.

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, 248–259.

- ”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.
- ”Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation”. DeepLearning 0.1. LISA Lab. Retrieved 31 August 2013.
- Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. ACM Computing Surveys, 37(4):316–344, 2005.doi:10.1145/1118890.1118892

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. 311–318. CiteSeerX 10.1.1.19.9416.

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. 2672–2680.