Code Generator for UI Design

Naveksha Sood
Graduate Student, Luddy SICE
Indiana University, Bloomington
soodn@iu.edu

Prateesh Reddy Patlolla
Graduate Student, Luddy SICE
Indiana University Bloomington
prpatlol@iu.edu

Rahul Shamdasani
Graduate Student, Luddy SICE
Indiana University Bloomington
rshamdas@iu.edu

ABSTRACT

Automation with the help of machine learning (ML) is making several jobs easier and several others obsolete. In this paper, we discuss ML/deep learning (DL) techniques to assist front-end software developers in converting a screenshot of graphic user interface (GUI) design into HTML code. The data pipeline utilizes pix2code dataset, containing pairs of GUI screenshot and bootstrap code, and models this data using convolutional and recurrent layers based neural network. The model is later validated using GUI screenshots, and the generated bootstrap code is post-processed using a compiling algorithm to convert it into HTML code. »Results«

1 MOTIVATION

Software developers are using artificial intelligence (AI) to automate the tasks that as little as a decade ago needed human intelligence and supervision. But can AI automate the tasks within the software development life-cycle? The phases of software development are often sequential where requirements become explorations, explorations become mock-ups and prototypes, that developers turn into the final product. These steps are tedious but at its core, each phase converts the representation of the software from one medium to another, adding a significant overhead to realise a plan. We can leverage the current state of the art ML and DL techniques to bring this testing time close to 0. In this study, we focus on the repetitive and time consuming task of generating the code for a GUI for multiple platforms. In essence, a GUI code is no different from a textual description of a screen that follows specific syntactic and semantic rules, and hence, the task of generating code for a GUI screenshot can be mapped to a task of image captioning.

The approach described in this study uses a convolutional neural network based model to extract the features from the pixel values of the screenshot, a recurrent neural network based model to extract the features of the coding/scripting language. The features of the GUI screenshot and the language are fed into another recurrent neural network based model to understand the mapping between the two. The models are trained using gradient descent algorithm based of cross-entropy loss. Towards the end of the paper, we present the qualitative and quantitative outcome of our experiments, which show promising results and can be scaled up to several other platforms.

2 RELATED WORK

The automation of the task of writing code has been underway since a long time, and we have seen progress in form of auto-complete, code-suggestion and drag-and-drop coding structures. In recent years, natural language processing (NLP) has paved its way to model coding and scripting languages like a natural language.

Reference [1] describes a pilot study on modelling programming languages as a natural language, and applied it to convert python2 code into python3 code. Reference [2] leveraged a high volume of freely available large code repositories online to build a gigatoken probabilistic language model of source code for Java language. Reference [3] built a system, Naturalize, and reference [4] build a similar model, that uses NLP to analyze the different coding style of developers and defines one standard coding convention for a project. In a similar study, Reference [5] and [6] uses recurrent neural networks to find and fix the syntactical errors in the code. Reference [7] and [8] is another application of NLP to mine code idioms which does more than code auto-complete and assists a developer to write code using libraries that he is rather unfamiliar with. Reference [9] and [10] build probabilistic models to convert natural language description into code and vice-versa. Reference [11] proposes an n-gram model for an application as a plagiarism detector by being able to differentiate between trivial and distinctive snippets of code in programs. Reference [12] introduces DeepSoft, a system that improves upon n-grams model for software code and uses long short-term memory model (LSTM) to model code that often has long-term dependencies.

One of the most popular applications in this domain is GitHub co-pilot [13]. Still in its beta testing phase, GitHub co-pilot is an AI pair programmer that helps write fragments of an algorithm or an entire algorithm, based on the comments and the context of the program. Another application DeepCoder [14] redefines the task of generating code for a given input and output as a search problem and tries to find the best match for the program from the plethora of already written programs.

While most of the prior art has focused on generating code from textual cues. Sketch2Code [15] and pix2code [16] claim to pioneer in the task of generating code of visual inputs. More specifically, sketch2code converts hand-drawn sketches into an intermediary code. Pix2code convert GUI screenshots to intermediary code using deep neural networks. In another contemporary art, AI Blueprint engine [17] generates deep neural networks from the visual input, moreover, it also creates an entire data pipeline for the project including python source code, requirements and package dependencies, docker files and a read-me for the project.

Our study builds upon the prior art, where we develop a convolutional and recurrent layers based neural network to generate bootstrap code for screenshots of GUI design, which is later converted to HTML code. The performance evaluation is carried out by finding loss between original bootstrap code and the generated bootstrap code, and visual difference in the HTML page corresponding to the two bootstrap files.

3 DATA

3.1 Dataset Overview

The dataset used for this study is a subset of open-source dataset, pix2code, containing GUI screenshot and code pairs for three different platforms, namely, web-based technologies, andriod and iOS. The distribution of data is described in the table 1 and figure 1.

Platform	Web	Android	iOS
All Pairs	1742	1749	1745
Train Set	1392	1399	1395
Eval Set	350	350	350

Table 1: Data Distribution of Pix2Code Dataset

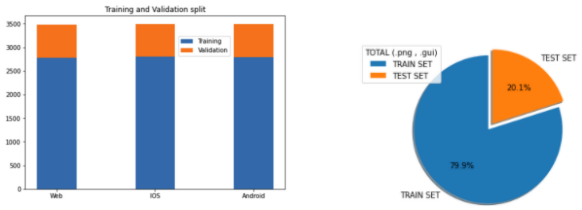

Figure 1: Data Distribution of Pix2code and Train-evaluation Split of Web Data

Of 3484 samples of web data, 700 samples (350 pairs of GUI screenshot and bootstrap code) are used as evaluation set and the remaining samples are used as training set. A sample of GUI screenshot and corresponding bootstrap code have been depicted in figure 2.

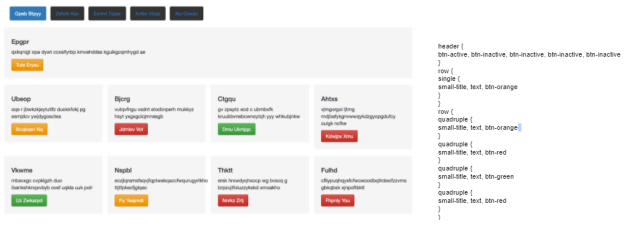

Figure 2: Data Sample from Pix2code Web Data

4 METHODOLOGY

The task of generating working HTML code from GUI screenshots was divided into three sub-problems. First sub-problem was a computer vision task which extracts features from a GUI screenshot given as an input to a convolutional neural network model. Second sub-problem was a language modelling task which extracts features from the code describing the GUI screenshot using a recurrent neural network. The features extracted from the two models were fed as input to another recurrent neural network which maps the elements of bootstrap code to the elements of GUI screenshot.

4.1 Computer Vision Model

Convolution Neural Network (CNN) model was used to extract features from the GUI screenshot and converting it into a feature vector. CNN follow a hierarchical model which builds a funnel of network, finally resulting in a fully connected layers along with their ability of feature parameter sharing and dimensionality reduction[15]. In our system, these CNN's work as an encoder to and convert the image into features, which makes this conversion general and not specific to web.

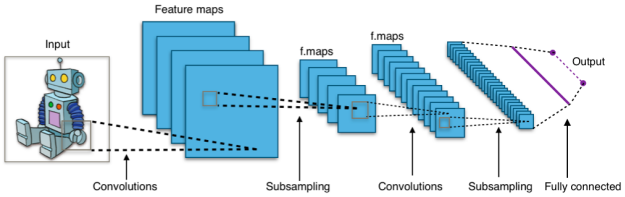

Figure 3: Image Model [17]

Before the training all the images were converted to a square image of 256 pixels and the values were normalized to make the training process more efficient.

4.2 Language Model

This language model is trained to establish relationships and hierarchical structure of a web page, it identifies the containers and categorize them as parent and child containers. However, the data in the web page is not of interest, we only model the bootstrap code which is descriptive enough to tell us the geographical position of the containers. It parses all the containers and gives a confidence score in the form of one hot encoded vector.

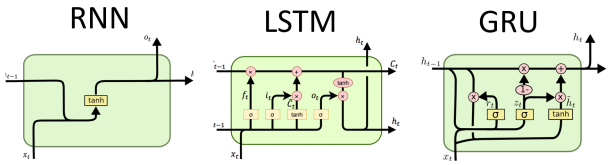

Figure 4: Language Model[16]

A recurrent neural network was the model of choice for this task. Since the number of children in the container are variable for every container and traditional RNN will fail in such a scenario so the best choice was LSTM which could keep track of closing tags important for our program syntactically correct for the compiler to interpret.

4.3 Combined Parser

For the final step we need to map the outputs of the two models and generate a syntactically correct HTML code. First input to this model is the output from the computer vision model which encodes the GUI image into a vector 'E', where E(i) is the i^{th} feature vector or the i^{th} container. Second input to this model will be the output of

language model which is also a vector 'F' signifying the relationship between tags, such that F(i) is the tag at i^{th} position. Main purpose of this model is to map these two vectors since the proper tags can be retained form first model currently stored in E and the corresponding geographic position are shown in the second vector F which is output from the language model. A supervised learning is employed on a recurrent neural network for this mapping.

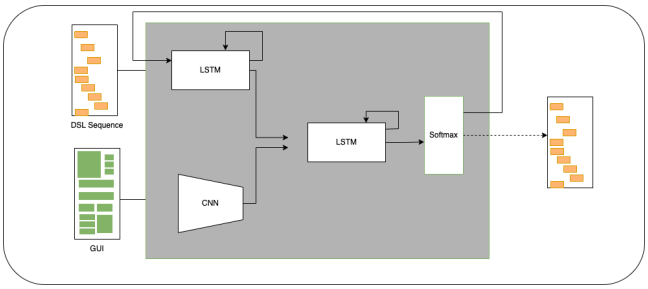

Figure 5: Combined Parser

5 EXPERIMENTS

The model as described above was an autoencoder and decoder based model, which employed convolutional neural network and recurrent neural network.

5.1 CNN - GRU

In the image model, we used eight CNN layers with small 3 × 3 receptive fields with every other layer followed by a max-pooling layer. The network has first two layers of width 16, followed by two layers of width 32, followed by two layers of width 64, followed by last two layers of width 128, followed two fully connected dense layers of width 1024.

The language model is two stacks of GRU layers, each of width 128.

5.2 CNN - LSTM

In the image model, we used 6 CNN layers with small 3 × 3 receptive fields with every other layer followed by a max-pooling layer. The network has first two layers of width 32, followed by 2 layers of width 64, followed by last 2 layers of width 128, followed 2 fully connected dense layers.

The language model is 2 stacks of LSTM layers, each of width 128.

5.3 CNN - BLSTM

In the image model, we used 6 CNN layers with small 3 × 3 receptive fields with every other layer followed by a max-pooling layer. The network has first two layers of width 32, followed by 2 layers of width 64, followed by last 2 layers of width 128, followed 2 fully connected dense layers.

The language model is 2 stacks of bidirectional LSTM (BLSTM) layers, each of width 128.

5.4 Combined Parser - LSTM

The combined parser model is 2 stacks of LSTM layers of width 512. We also experimented with BLSTM layers in place of LSTM.

The activation function used was Rectified Linear Unit (ReLU). The optimizers that we tried were RMSProp, Adam and SGD, where RMSProp worked the best. Dropout was added after experimentation of each model that seemed to overfit. The learning rate was varied between 0.01 to 0.0001 in steps. The performance metric used was categorical cross entropy loss for weight training. And model was allowed to train in batches of 64 for 200 epochs which took roughly 20 hours for each model, with early stopping in place.

5.5 Sampling

The input image to the vision model is used to predict a token based on the previously generated 47 tokens. Initially all the tokens are set to empty, except for the first token which is set to <START> signifying the start of code, at the first iteration of training the first 48 tokens are manipulated, in the next iteration we discard the first token and tokens from two to 49 are manipulated, in this scenario each token is predicted approximately 48 times and the argmax is taken to decide the best value of this token.

When we reach a new state, i.e. when we move from predicting 48^{th} token to predicting the 49^{th} token, we experiment with the following two techniques to identify the best possible fit for this token. First, the greedy approach, where we simply explore the most promising state at each level and generate a sequence based only on those states. Second, the beam search technique in which we explore n promising states where n is the beam length. We found the beam search with length 3 performed better than the greedy approach.

5.6 Post-processing of Generated Code

The bootstrap code generated from the model needs to be verified for correctness, and how close it is to the expected output. To do this, a compilation algorithm is used that converts the bootstrap into the HTML code with random text, since text is not of importance here.

5.7 Performance Metrics

While it was easy to evaluate the results of the model qualitatively by examining the similarities of the ground truth and generated web page. We tried a number of quantitative measures to evaluate model performance like categorical cross entropy loss, bilingual evaluation understudy (BLEU) score, perplexity, pixel-by-pixel matching of the ground truth and generated image. Of these performance metrics, BLEU score seemed to be the best reflector of the model's performance.

6 RESULTS

Of the 4 model architectures that we tried, we found VGGNet as image model and LSTM based language model to be performing the best. A sample of ground truth and generated HTML page by the model in the figures below.

The model resulted in the BLEU of 0.78 for 350 test pairs and gave promising qualitative result for the real life application of our algorithm.

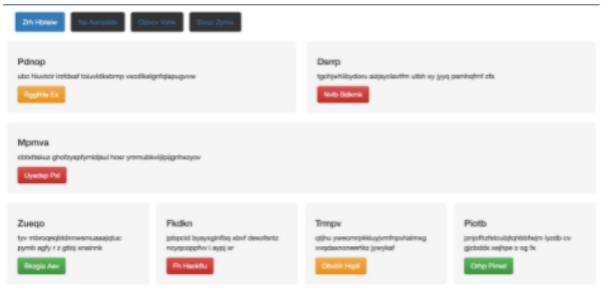

Figure 6: Ground Truth

Figure 7: Generated Image

7 CONCLUSION AND FUTURE SCOPE

Auto-code generation not only reduces developers efforts but also reduces down the time to visualize a plan significantly. Various researchers have worked on modelling coding languages for applications like automatic code generation, fixing syntactical errors, detecting plagiarism efficiently, automatic code documentation etc. but only a handful number of studies have explored the area to generate code from visual cues. Our study aimed to tackle this problem using deep learning techniques, and with thorough experimentation defined in the report we hope to set a benchmark in the domain. The task of code generation for GUI screenshot is accomplished by using 3 models, one to convert the image to a feature vector using convolutional neural network, second to convert the code to a feature vector using LSTM based neural network and the final model to find the mapping of the output of the previous two models to each other. Having learnt the mapping between the image vector and the language vector, our model can now generate bootstrap code for any GUI screenshot it gets as input in the validation phase. The bootstrap code is converted to HTML code using a compilation algorithm. In the validation phase, the generated code and ground truth code are matched using BLEU score. Our approach resulted in 0.78 BLEU score and shows promising qualitative results.

We hope to extend our study on various platforms, and test the existing model on hand drawn sketches as well. A rather ambitious goal that we will work towards is generating interlinked web pages using UI/UX code from Adobe XD.

8 REFERENCES

(1) Aggarwal K, Salameh M, Hindle A. 2015. Using machine translation for converting Python 2 to Python 3 code. PeerJ PrePrints 3:e1459v1

(2) M. Allamanis and C. Sutton, "Mining source code repositories at massive scale using language modeling," 2013 10th Working Conference on Mining Software Repositories (MSR), 2013, pp. 207-216, doi: 10.1109/MSR.2013.6624029.

(3) Miltiadis Allamanis, Earl T. Barr, Christian Bird, and Charles Sutton. 2014. Learning natural coding conventions. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014). Association for Computing Machinery, New York, NY, USA, 281–293. DOI:https://doi.org/10.1145/2635868.2635883

(4) Hellendoorn, P. T. Devanbu and A. Bacchelli, "Will They Like This? Evaluating Code Contributions with Language Models," 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015, pp. 157-167, doi: 10.1109/MSR.2015.22.

(5) Bhatia S., Singh R., "Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks," 2016 arxiv.org/abs/1603.06129

(6) R. Corchuelo, J. A. Pérez, A. Ruiz, and M. Toro. Repairing syntax errors in LR parsers. ACM Trans. Program. Lang. Syst., 24(6):698–710, Nov. 2002.

(7) Bhoopchand A., Rocktäschel T., Barr E., Riedel S., "Learning Python Code Suggestion with a Sparse Pointer Network," 2016 1611.08307

(8) Miltiadis A.,Charles S., "Mining idioms from source code",2014 https://arxiv.org/abs/1404.0417

(9) Miltiadis Allamanis, Daniel Tarlow, Andrew D. Gordon, and Yi Wei. 2015. Bimodal modelling of source code and natural language. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15). JMLR.org, 2123–2132.

(10) Barone M., Valerio A., Rico S., "A Parallel Corpus of Python Functions and Documentation Strings for Automated Code Documentation and Code Generation," 2017 https://aclanthology.org/I17-2053

(11) https://arxiv.org/pdf/1705.07962v1.pdf

(12) Dam, Khanh Hoa, T. Tran and Trang Pham. "A deep language model for software code." ArXiv abs/1608.02715 (2016): n. pag.

(13) https://copilot.github.com/

(14) https://techcrunch.com/2017/02/23/deepcoder-builds-programs-using-code-it-finds-lying-around/

(15) https://arxiv.org/pdf/1910.08930.pdf

(16) https://arxiv.org/pdf/1705.07962v1.pdf

(17) https://medium.com/@creaidAI/introducing-the-ai-blueprint-engine-a-code-generator-for-deep-learning-31093499b246

(18) https://intellipaat.com/community/46830/why-convolutional-neural-network-is-better

(19) http://dprogrammer.org/rnn-lstm-gru

(20) https://commons.wikimedia.org/wiki/File:Typical$_c$nn.png

Code : *https : //github.com/navekshasood/deep_learning*