JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

# Vision2Code: Transformation of Sketches to UI in Real-time Using Deep Neural Network

**[1]Khushbu S. Birhade, [2]Saurabh Bansod**
[1]Student at National Institute Electronics and Information Technology, Aurangabad,
[2]Mtech Coordinator at National Institute Electronics and Information Technology, Aurangabad

***Abstract:*** In a Development Work Most of The Time we are spending our most of the precious time in developing UI prototypes and this can be tedious work, and it leaves very smaller window for building the actual logic and functionality for the product which alternatively affects the productivity. User Interface (UI) prototyping is a necessary step in the early stages of application development. Transforming sketches of a Graphical User Interface (UI) into a coded UI application is an uninspired but time-consuming task performed by a UI designer .Just imagine that the time that we are giving for developing The UI prototype, if we give the similar amount of time to make the system more robust and more functional then , how much efficient our product would be ? So to find the alternative of the above problem I am proposing the solution where we do not require to think about the HTML code for developing the UI, it will be completely automatic using the current advancement into the technology called Deep Neural network .Here You just need to draw a design ( wireframe ) on a white board or white paper and then using the Computer vision it will automatically generate the HTML output code for it . This could Help the developers to work efficiently and also it can save a lot of time of the employee and the company and saving the time of the company alternatively saves the money. The output of the project would be the code which can further modified according to the user requirement. The two main models which will plays main role here are convolution neural network and gated recurrent network. Here the model will be trained on the data which I will generate by me. Where I need to take the picture of the hand draw designs. And a last for the deployment point of you the webapp will be created which will give the user more user friendly environment where they just need to take the picture or they just need to upload it and the output will be printed of screen.

***IndexTerms* – Computer, CNN, deep neural network , GUI.**

## I. Introduction

The process of implementing client-side software supported a Graphical programmer (GUI) mockup created by a designer is that the responsibility of developers. Implementing GUI code is, however, time-consuming and stop developers from dedicating the bulk of their time implementing the actual functionality and logic of the software they're building. Moreover, the pc languages used to implement such GUIs are specific to every target runtime system; thus leading to tedious and repetitive work when the software being built is predicted to run on multiple platforms using native technologies. During this paper, we describe a model trained end-to-end with stochastic gradient descent to simultaneously learn to model sequences and spatio-temporal visual features to come up with variable-length strings of tokens from one GUI image as input. Our first contribution is pix2code, a unique approach supported Convolutional and Recurrent Neural Networks allowing the generation of computer tokens from one GUI screenshot as input. That is, no engineered feature extraction pipeline nor expert heuristics was designed to process the input data; our model learns from the pixel values of the input image alone. Our experiments demonstrate the effectiveness of our method for generating coding system for various platforms (i.e. iOS and Android native mobile interfaces, and multi-platform web-based HTML/CSS interfaces) without the need for any change or specific tuning to the model. In fact, pix2code may be used intrinsically to support different target languages just by being trained on a distinct dataset. A video demonstrating our system is on the market online1 . Our second contribution is that the release of our synthesized datasets consisting of both GUI screenshots and associated ASCII text file for 3 different platforms. Our datasets and our pix2code implementation are publicly available2 to foster future research. The very beginning in creating an application or website is to sketch a wireframe or to create a graphical user interface screenshot. Designers face a challenge when converting their wireframe or GUI into code, this work often consumes time for the developer and so increases the price. The process of implementing a Graphical computer programmer (GUI) mockup created by a designer and converting directly into a web site is that the responsibility of developers. Most present-day user-facing software programming applications are Graphical computer program (GUI) driven, and depend on alluring computer programmed (UI). But implementing GUI code is, however, time-consuming and prevents developers from dedicating the bulk of their time to implementing the particular functionality and logic of the software they're building. The project is from one GUI image as input to get computer UI code, using Deep Learning Techniques. to coach the model on different data sets for various effective output codes. we would like to make a neural network that may be generating HTML/CSS markup that corresponds to a user's screenshot. When you train the neural network, you provides it many screenshots with matching HTML. It learns by predicting the matching HTML markup tags one after the another. When it predicts the next markup tag, it receives the screenshot and every one the correct markup tags until that time

## RELATED WORK

Present-day, front-end developers work so hard for a perfect GUI interface. Avoid the frustrations that front-end engineers and designers experience when creating precise GUIs; this highlights the requirement for more practical resolution in website design. During this study, to automatically produce the HTML code for the mockup of an internet site page an approach has been created. It's planned to recognize the elements created within the mock-up images and encode them in accordance with the online site page hierarchy. To train, the deep neural network model, which has Convolution Neural Networks (CNN), is employed to teach the images present on the data sets.

A recent example is pix2code, an approach supported Convolutional and Recurrent Neural Networks which allows the assembly of computer tokens from one Graphical package screenshot as input but their Model was trained on a relatively small dataset, hence accuracy was less. Whereas, Sketch2code used classical wireframe techniques and deep learning techniques code was generated by pre-processing and segmentation but it also has moderate results since input relies on the camera of the device. Another example is Deep Coder, a system ready to generate computer programs by leveraging statistical predictions to augment traditional search techniques.

The technique reduces the runtime of an oversized kind of IPS baselines by 1-3 orders of magnitude. Several problems in real online programming challenges that will be solved with a program during this proposed language. But the problems which are solved by the model are relatively simpler than any other competitive exams. The input-output examples become less informative because the DSL becomes more complex.

In this paper, Graves explains how to use a Recurrent Neural Network to approximate probability distribution function (PDF). The paper proposes using LSTM cells in order to educate the RNN to remember information from the distant past. There with modification, the PDF of the subsequent value of the sequence could also be approximated as a function of the current value of the sequence and also the worth of the RNN's current hidden state. The model fails to educate on more complicated datasets, thanks to the more complex and diverse nature of each image.

In Improving pix2code-based Bi-directional LSTM by Yanbin Liu, Qidi Hu, Kunxian Shu where model is optimized by Bidirectional LSTM, which allows the output layer to urge complete context of past and future information for each point within the input sequence.. Decoder exploits the advantages of CNN in feature extraction combined with the advantages of BLSTM in processing sequence problems to automatically generate code. The model's transforming accuracy within the test set has been significantly improved reaching 85%. But since, The model uses Bidirectional LSTM, hence the training time of the model is more.

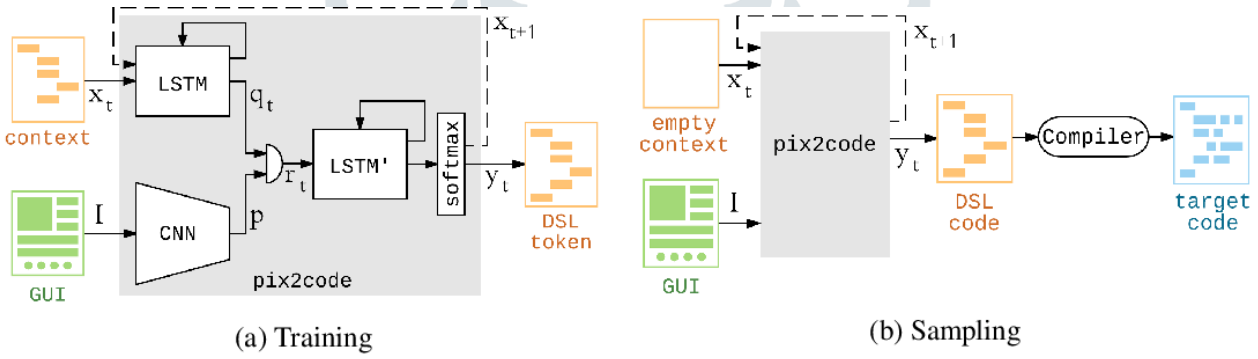
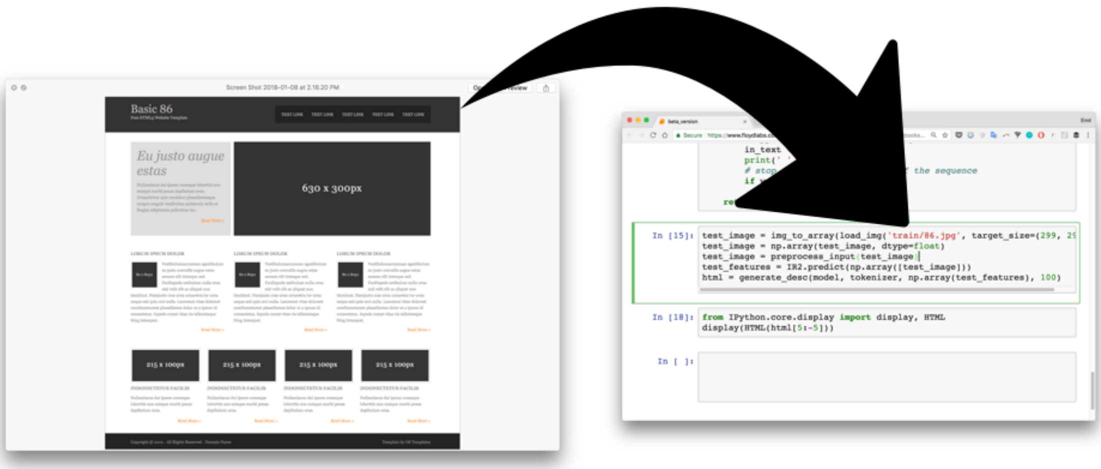


Fig.1: CNN based vision model

During training, the GUI image is encoded by a CNN-based vision model; the context (i.e. a sequence of one-hot encoded tokens admire DSL code) is encoded by a language model consisting of a stack of LSTM layers. The 2 resulting feature vectors are concatenated and fed into the second stack of LSTM layers acting as a decoder. Finally, a softmax layer is employed to sample one token at a time; the output size of the softmax layer corresponds to the DSL vocabulary size. Given a picture and a sequence of tokens, the model (i.e.contained within the gray box) is differentiable and may thus be optimized end-to-end through gradient descent to predict the subsequent token within the sequence. During sampling, each prediction's input context is updated to contain the last predicted token. The resulting sequence of DSL tokens is compiled to the specified target language using traditional compiler design techniques.

### Vision Model

CNNs are currently the tactic of option to solve a good range of vision problems because of their topology allowing them to be told rich latent representations from the pictures they're trained on [16, 11]. We used a CNN to perform unsupervised feature learning by mapping an input image to a learned fixed-length vector; thus acting as an encoder as shown in Figure 1. The input images are initially re-sized to 256 × 256 pixels (the ratio isn't preserved) and also the pixel values are normalized before to be fed within the CNN. No further pre-processing is performed. To encode each input image to a fixed-size output vector, we exclusively used small 3 × 3 receptive.



```
        in_text
        print('
        # stop                         the sequence
        if

    re

In [15]: test_image = img_to_array(load_img('train/86.jpg', target_size=(299, 29
         test_image = np.array(test_image, dtype=float)
         test_image = preprocess_input(test_image)
         test_features = IR2.predict(np.array([test_image]))
         html = generate_desc(model, tokenizer, np.array(test_features), 100)

In [18]: from IPython.core.display import display, HTML
         display(HTML(html[5:-5]))

In [ ]:
```

Figure 2: design image to the trained neural network.

**LANGUAGE MODEL**

We designed an easy lightweight DSL to explain GUIs as illustrated in Figure 2. during this work we are only fascinated by the GUI layout, the various graphical components, and their relationships; thus the particular textual value of the labels is ignored. Additionally to reducing the dimensions of the search space, the DSL simplicity also reduces the scale of the vocabulary (i.e. the full number of tokens supported by the DSL). As a result, our language model can perform token-level language modeling with a discrete input by using one-hot encoded vectors; eliminating the necessity for word embedding techniques like word2vec [13] which will end in costly computations. In most programming languages and markup languages, a component is said with a gap token; if children elements or instructions are contained within a block, a closing token is typically needed for the interpreter or the compiler. In such a scenario where the amount of youngsters elements contained during a parent element is variable, it's important to model long-term dependencies to be ready to close a block that has been opened. Traditional RNN architectures suffer from vanishing and exploding gradients preventing them from having the ability to model such relationships between data points detached in statistic (i.e. during this case tokens unfolded in an exceedingly sequence). Hochreiter and Schmidhuber proposed the Long STM (LSTM) neural architecture so as to handle this very problem [9]. the various LSTM gate outputs may be computed as follows:

it = _(Wixxt +Wiyht1 + bi) (1)
ft = _(Wfxxt +Wfyht1 + bf ) (2)
ot = _(Woxxt +Woyht1 + bo) (3)
ct = ft _ ct1 + it _ _(Wcxxt +Wcyht1 + bc) (4)
ht = ot _ _(ct) (5)

With W the matrices of weights, at the new input vector at time t, ht1 the previously produced output vector, ct1 the previously produced cell state's output, b the biases, and that i and that i the activation functions sigmoid and hyperbolic tangent, respectively. The cell state c learns to memorize information by employing a recursive connection as wiped out traditional RNN cells. The input gate i is employed to controlling the error flow on the inputs of cell state c to avoid input weight conflicts that occur in traditional RNN because the identical weight must be used for both storing certain inputs and ignoring others. The output gate o controls the error be due the outputs of the cell state c to forestall output weight conflicts that happen in standard RNN because the identical weight needs to be used for both retrieving information and not retrieving others. The LSTM memory block can thus use I to choose when to jot down information in c and use o to make a decision when to read information from c. We used the LSTM variant proposed by Gers and Schmidhuber with a forget gate f to reset memory and help the network model continuous sequences.

**3.3 Decoder**

Our model is trained in an exceedingly supervised learning manner by feeding a picture I and a contextual sequence X of T tokens xt; t 2 f0: T 1g as inputs; and therefore the token xT because the target label. As shown in Figure 1, a CNN-based vision model encodes the input image I into a vectorial representation p. The input token xt is encoded by an LSTM-based language model into an intermediary representation qt allowing the model to focus more on certain tokens and fewer on others [8]. This mother tongue model is implemented as a stack of two LSTM layers with 128 cells each. The vision-encoded vector p and therefore the language-encoded vector qt are concatenated into one feature vector rt which is then fed into a second LSTM-based model decoding the representations learned by both the vision model and therefore the language model. The decoder thus learns to model the link between objects present within the input GUI image and therefore the associated tokens present within the DSL code. Our decoder is implemented as a stack of two LSTM layers with 512 cells each. This architecture will be expressed mathematically as follows:

p = CNN(I) (6)
qt = LSTM(xt) (7)
rt = (q; pt) (8)
yt = softmax(LSTM0(rt)) (9)
xt+1 = yt (10)

This architecture allows the entire pix2code model to be optimized end-to-end with gradient descent to predict a token at a time after it's seen both the image similarly because the preceding tokens within the sequence. The discrete nature of the output (i.e. fixed-sized vocabulary of tokens within the DSL) allows us to cut back the task to a classification problem. That is, the output layer of our model has the identical number of cells because of the vocabulary size; thus generating a probability distribution of the candidate tokens at when step allowing the employment of a softmax layer to perform multi-class classification.

| Dataset type | Error (%) | | |
|---|---|---|---|
| | greedy search | beam search 3 | beam search 5 |
| iOS UI (Storyboard) | **22.73** | 25.22 | 23.94 |
| Android UI (XML) | **22.34** | 23.58 | 40.93 |
| web-based UI (HTML/CSS) | 12.14 | **11.01** | 22.35 |

### 3.4 Training

The length T of the sequences used for training is important to model long-term dependencies; for example to be able to close a block of code that has been opened. After conducting empirical experiments, the DSL input files used for training were segmented with a sliding window of size ; in other words, we unroll the recurrent neural network for 48 steps. This was found to be a satisfactory trade-off between long-term dependencies learning and computational cost. For every token in the input DSL file, the model is therefore fed with both an input image and a contextual sequence of T = 48 tokens. While the context (i.e. sequence of tokens) used for training is updated at each time step (i.e. each token) by sliding the window, the very same input image I is reused for samples associated with the same GUI. The special tokens < START > and < END > are used to respectively prefix and suffix the DSL files similarly to the method used by Karpathy and Fei-Fei . Training is performed by computing the partial derivatives of the loss with respect to the network weights calculated with backpropagation to minimize the multiclass log loss: Sampling To generate DSL code, we feed the GUI image I and a contextual sequence X of T = 48 tokens where tokens xt : xT1 are initially set empty and the last token of the sequence xT is set to the special < START > token. The predicted token yt is then used to update the next sequence of contextual tokens. That is, xt :: xT1 are set to xt+1 : xT (xt is thus discarded), with xT set to yt. The process is repeated until the token < END > is generated by the model. The generated DSL token sequence can then be compiled with traditional compilation methods to the desired target language. (a) pix2code training loss (b) Micro-average ROC curves Figure 3: Training loss on different datasets and ROC curves calculated during sampling with the
model trained for 10 epochs.

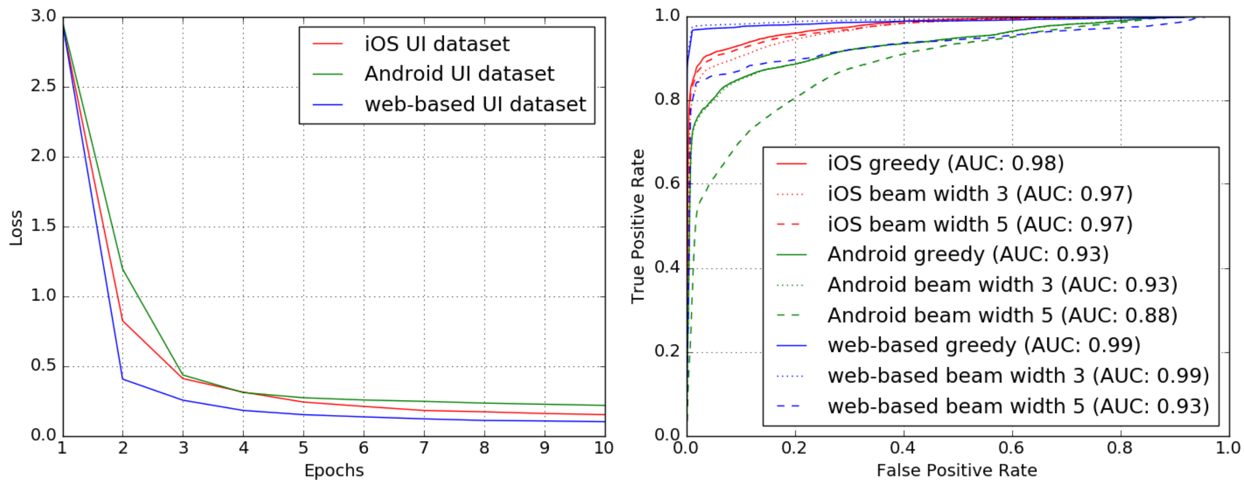


Figure 3: Training loss on different datasets and ROC curves calculated during sampling with the model trained for 10 epochs.

### EXPERIMENTS

Access to consequent datasets could be a typical bottleneck when training deep neural networks. To the simplest of our knowledge, no dataset consisting of both GUI screenshots and ASCII text file was available at the time this paper was written. As a consequence, we synthesized our own data leading to the three datasets described in Table 1. The column Synthesizable refers to the utmost number of unique GUI configurations which will be synthesized using our stochastic computer programmed generator. The columns Instances discuss with the amount of synthesized (GUI screenshot, GUI code) file pairs. The column Samples ask the amount of distinct image-sequence pairs. In fact, training and sampling are done one token at a time by feeding the model with a picture and a sequence of tokens obtained with a window of fixed size T. the entire number of coaching samples thus depends on the overall number of tokens written within the DSL files and also the size of the window. Our stochastic computer programmed generator is meant to synthesize GUIs written in our DSL which are then compiled to the required target language to be rendered. Using data synthesis also allows us to demonstrate the potential of our model to come up with code for 3 different platforms. Our model has around 109 _ 106 parameters to optimize and every one experiments are performed with the identical model with no specific tuning; Code generation is performed with both greedy search and beam search to seek out the tokens that maximize the classification probability. To gauge the standard of the generated output, the classification error is computed for every sampled DSL token and averaged over the entire test dataset. The length difference between the generated and also the expected token sequences is additionally counted as a slip. The results will be seen in Figures 4 show Experimental samples for the IOS GUI dataset. Consisting of input GUIs (i.e. ground truth), and output GUIs generated by a trained pix2code model. It’s important to recollect that the particular textual value of the labels is ignored which both our data synthesis algorithm and our DSL compiler assign randomly generated text to the labels. Despite occasional problems to pick the correct color or the proper style for specific GUI elements and a few difficulties modeling GUIs consisting of long lists of graphical components, our model is mostly able to learn the GUI layout in an exceedingly satisfying manner and may preserve the hierarchical data structure of the graphical elements.

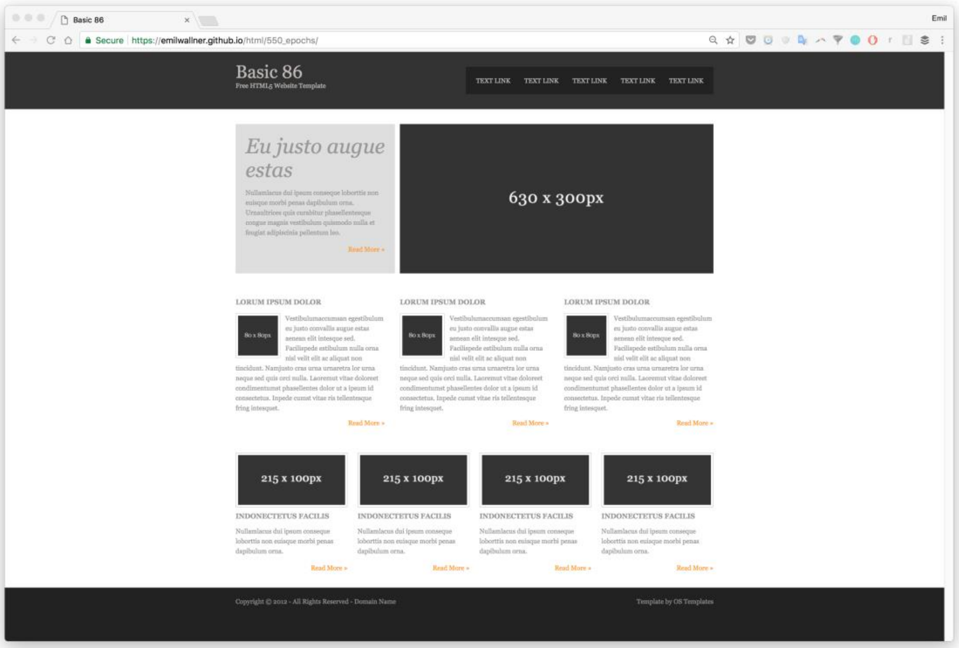


Figure 4: Experiment samples for the IOS GUI dataset.

## CONCLUSION AND DISCUSSIONS

In this paper, we presented vision2code, a completely unique method to come up with code given one GUI image as input. While our work demonstrates the potential of such a system to automate the method of implementing GUIs, we only scratched the surface of what's possible. Our model consists of relatively few parameters and was trained on a comparatively small dataset. the standard of the generated code can be drastically improved by training an even bigger model on significantly more data for an extended number of epochs. Implementing a now-standard attention mechanism [1, 22] could further improve the standard of the generated code. Using one-hot encoding doesn't provide any useful information about the relationships between the tokens since the strategy simply assigns an arbitrary victoria representation to every token. Therefore, pre-training the language model to find out victoria representations would allow the relationships between tokens within the DSL to be inferred (i.e. learning word embedding like word2vec [13]) and as a result alleviate semantically error within the generated code. Furthermore, one-hot encoding doesn't scale to very big vocabulary and thus restricts the amount of symbols that the DSL can support. Figure 5: Experiment samples from the web-based GUI dataset. Generative Adversarial Networks GANs [7] have shown to be extremely powerful at generating images and sequences [23, 15, 25, 17, 3]. Applying such techniques to the matter of generating coding system from an input image is up to now an unexplored research area. GANs could potentially be used as a standalone method to come up with code or may well be employed in combination with our pix2code model to fine-tune results. a serious drawback of deep neural networks is that the need for lots of coaching data for the resulting model to generalize well on new unseen examples. one in all the many advantages of the tactic we described during this paper is that there's no need for human-labeled data. In fact, the network can model the relationships between graphical components and associated tokens by simply being trained on image-sequence pairs. Although we used data synthesis in our paper partly to demonstrate the potential of our method to get GUI code for various platforms; data synthesis won't be needed in any respect if one wants to focus only on web-based GUIs. In fact, one could imagine crawling the planet Wide Web to gather a dataset of HTML/CSS code related to screenshots of rendered websites. Considering an oversized number of sites already available online and also the proven fact that new websites are created each day, the online could theoretically supply a virtually unlimited amount of coaching data; potentially allowing deep learning methods to completely automate the implementation of web-based GUIs.

## REFERENCES

[1] Tony Beltramelli UIzard Technologies Copenhagen, Denmark, “pix2code: Generating Code from a Graphical User Interface Screenshot” Proc.ACM SIGCHI Symp. Eng. Interact. Comput. Syst, June2017.
[2] Alexander Robinson, University of Bristol, “sketch2code: Generating a website from a paper mockup,” Department of Computer Science, May 2018.
[3] Matej Balog, Department of Engineering University of Cambridge and Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, Daniel Tarlow, Microsoft Research, “DEEPCODER: LEARNING TO WRITE PROGRAMS,” ICLR 2017.
[4] Yanbin Liu, Qidi Hu, Kunxian Shu, “Improving pix2code based Bi-directional LSTM” IEEE International Conference on Automation, Electronics and Electrical Engineering, 2018.
[5] Alex Graves.”Generating Sequences With Recurrent Neural Networks” Department of Computer Science University of Toronto, June 2014
[6] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

[7] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. Deepcoder: Learning to write programs. arXiv preprint arXiv:1611.01989, 2016.
[8] B. Dai, D. Lin, R. Urtasun, and S. Fidler. Towards diverse and natural image descriptions via a conditional gan. arXiv preprint arXiv:1703.06029, 2017.
[9] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2625–2634, 2015.
[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] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. Neural computation, 12(10):2451–2471, 2000.
[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Advances in neural information processing systems, pages 2672–2680, 2014.
[13] A. Graves. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850, 2013.
[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural computation, 9(8):1735– 1780, 1997.
[15] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3128–3137, 2015.
[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.
[17] 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. 8
[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 3111–3119, 2013.
[19] T. A. Nguyen and C. Csallner. Reverse engineering mobile application user interfaces with remaui (t). In Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, pages 248–259. IEEE, 2015.
[20] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In Proceedings of The 33rd International Conference on Machine Learning, volume 3, 2016.
[21] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229, 2013.
[22] R. Shetty, M. Rohrbach, L. A. Hendricks, M. Fritz, and B. Schiele. Speaking the same language: Matching machine to human captions by adversarial training. arXiv preprint arXiv:1703.10476, 2017.
[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
[24] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1):1929–1958, 2014.