Generating Webpages from Screenshots

Andrew S. Lee[*]
School of Engineering
Stanford University
andrewslee@stanford.edu

Abstract

This project created a PyTorch implementation of an image-captioning model in order to convert screenshots of webpages into code, following pix2code[1]. The system passes images into a ResNet-152-based CNN encoder model, which generates features for a decoder RNN model which uses word embeddings and an LSTM. The project resulted in peak BLEU scores plateauing around 0.92 after a few hundred epochs.

1 Introduction

Over the past year, there has been a growing interest in applying deep learning methods to front-end development, such as AirBnb's work on "Sketching Interfaces"[2] and Tony Beltramelli's pix2code [1]. The potential of applying deep learning to this field is to better automate the creation of code based on mockups from designers. Today, this is often done algorithmically and without much flexibility via offerings like SquareSpace or WebFlow.

This paper discusses replicating recent work done in transforming website GUI screenshots into code in PyTorch. The input to the model is a GUI screenshot of a Bootstrap-CSS-based website. We use an encoder CNN based on ResNet-152 to extract features from the image, and then a decoder RNN which uses word embeddings and an LSTM, in order to output a sequence of predicted DSL tokens. The result is a system which mimics existing image-captioning models, with a peak BLEU score of 0.92.

2 Related work

2.1 State of the Art: ReDraw and REMAUI

The current state-of-the-art in this area is a recent work (pre-published at the time of writing), *Machine Learning-based Prototyping* by Moran et al [5]. It proposes a novel technique of pre-processing GUI screen-shots into simpler components before training a deep neural network to classify those components. They then use a KNN-based algorithm to recreate hierarchies of code based on training from a diversified training set. Their approach is successful at converting arbitrary screenshots of Android applications into hierarchical layout descriptions and basic rendering code, which is much more advanced than the method used in our paper due to it avoiding the use of a DSL and its broader applicability to industrial uses. This approach is most similar to *REMAUI*, which uses OCR, CV, and heuristics to generate static applications.

[*]See: https://andrewlee.design/
[2]See: *Sketching Interfaces*.

Rewire describes a related problem of transforming screenshots into vector representations [8]. While not a deep-learning based solution, it uses existing OCR libraries and "an Ultra Metric Contour Map (UCM)" in order to segment images into vectorizable forms.

2.2 Our Basis: pix2code

pix2code is the primary work this paper is based off of. It is a keras-based implementation, which at its release also included the dataset used in this paper. The main difference between pix2code and our approach is that it is uses a single end-to-end model without a pre-trained CNN. This approach, at its face, seems like it would do better at the task because it uses its own CNN trained on interface images. However, its average accuracy of 77% is lower than our approach and Emil Wallner's (see next).

Another related work is a project developed by Emil Wallner, which is another keras implementation of pix2code, using the same dataset. It differentiates itself from *pix2code* by splitting the model into an encoder and decoder, and using a pre-trained CNN as the base of the encoder model.

3 Dataset and Features

The dataset we used maps Bootstrap-based[3] websites into a DSL consisting of 18 vocabulary tokens. The pix2code dataset consists of 3500 image and DSL markup pairs, which was split into 80% train, 10% cross-validation, and 10% test sets [1] . The dataset is rescaled by PyTorch from 2400 x 1380px to 224 x 224px in order to fit the requirements of our pre-trained ResNet-152 model.

The ResNet model is used to extract features from the image (size 1x1x2048 per screenshot) which we pass onto a decoder model.

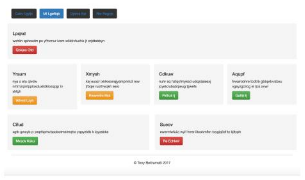
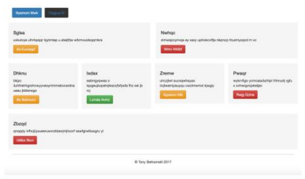
	
(a) header { btn-inactive, btn-inactive, btn-inactive, btn-active } row { quadruple { small-title, text, btn-red } quadruple { small-title, text, btn-red } quadruple { small-title, text, btn-green } quadruple { small-title, text, btn-red } } row { single { small-title, text, btn-green } } row { double { small-title, text, btn-red } double { small-title, text, btn-red } }	(b) header { btn-active, btn-inactive } row { double { small-title, text, btn-orange } double { small-title, text, btn-red } } row { quadruple { small-title, text, btn-orange } quadruple { small-title, text, btn-green } quadruple { small-title, text, btn-orange } quadruple { small-title, text, btn-red } } row { single { small-title, text, btn-red } }

Table 1: Examples of pix2code data (screenshot .png file and .gui text)

4 Methods

4.1 PyTorch

The approach used in this paper is based on PyTorch, meaning we take advantage of the framework's simpler abstractions [7]. This means we created a custom PyTorch DataLoader (and collation function) for the pix2code website dataset that lets us iterate through batches of image-caption pairs when training and testing.

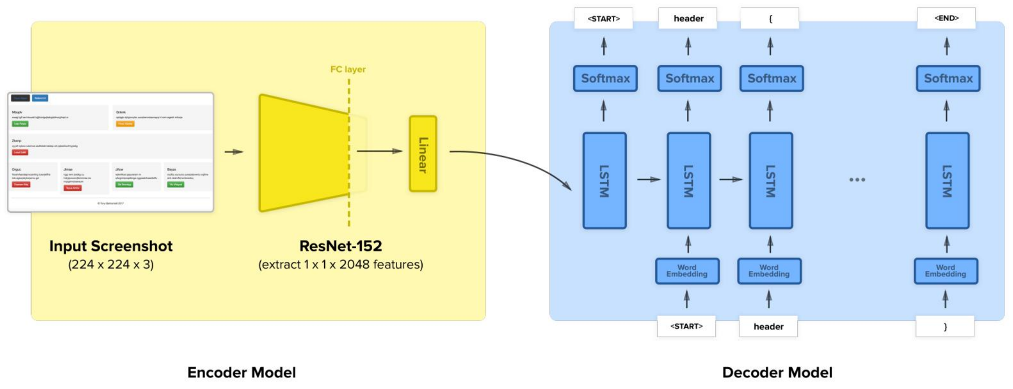


Figure 1: Our encoder and decoder models.

4.2 Models

Unlike the single end-to-end pix2code model, our system follows an image captioning model previously created for PyTorch[4]: an encoder CNN and a decoder RNN. As a whole, the system takes in a screenshot as input, and outputs a sequence of indices (based on the DSL language's vocabulary) which we then convert into valid HTML.

$$-(y \log(p) + (1-y)\log(1-p)) \tag{1}$$

The system uses an Adam optimizer for both the encoder and decoder [4], and a softmax cross entropy loss function, which provides probability values between 0 and 1 for a series of potential classes (Eqn. 1).

4.2.1 Encoder

Our encoder begins with a pre-trained ResNet-152 model as a base for transfer learning. This is a deep residual network which has shown increased accuracy at image classification based on the extreme depth of the network [2]. The primary change we made was to remove the final collection layer of the existing model in order to intercept detected features. Instead, we pass our data through a Linear PyTorch layer before moving to the decoder.

4.2.2 Decoder

The decoder takes in the expected target for the screenshot and the features from the encoder. It stores the target (a "caption") in a word embedding (using a PyTorch module which stores word embeddings via indices). Our source and target sequences, along with our our feature vector, is then used to train a Long Short Term Memory (LSTM) layer. An LSTM is a special kind of recurrent neural network which has the ability to learn and remember information overtime [3]. Our decoder's LSTM is what we teach a language model to based on the features it receives from the encoder.

5 Experiments/Results/Discussion

5.1 (Hyper)parameters

The main parameters we changed in our (limited) experimenting included: embedding size, number of epochs, and the hidden size. These parameters were chosen due to constraints in time while experimenting (as training in parallel was not possible).

[3]See Bootstrap.

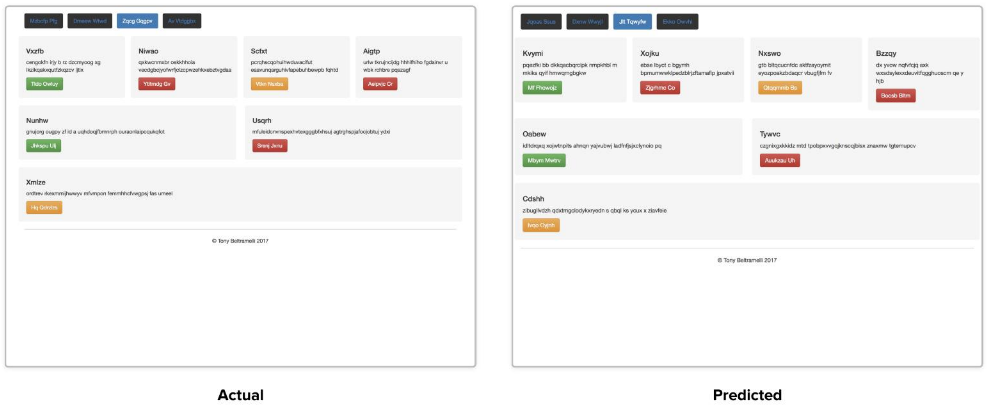
[4]See "Image Captioning" on GitHub.

We used 0.001 as our learning rate (which was our base choice). Our mini-batch size was 4, which was chosen after small experimentation showed this led to the fastest training.

5.2 BLEU Scores

We are using Bilingual Evaluation Understudy Scores (BLEU) to quantify our results, which is common for image-captioning models [6]. This is a score from 0.0 to 1.0, indicating how similar two sequences of tokens are (where 1.0 is the highest).

5.3 Results



header { btn-inactive, btn-inactive, btn-active, btn-inactive } row { quadruple { small-title, text, btn-green } quadruple { small-title, text, btn-red } quadruple { small-title, text, btn-orange } quadruple { small-title, text, btn-red } } row { double { small-title, text, btn-green } double { small-title, text, btn-red } } row { single { small-title, text, btn-orange } }

header { btn-inactive , btn-inactive , btn-active , btn-inactive } row { quadruple { small-title , text , btn-green } quadruple { small-title , text , btn-red } quadruple { small-title , text , btn-orange } quadruple { small-title , text , btn-red } } row { double { small-title , text , btn-green } double { small-title , text , btn-red } } row { single { small-title , text , btn-orange } }

Figure 2: Our model's predicted website compared to the target.

After training our encoder and decoder models over a number of epochs, we would use a separate sampling function to iteratively (100 iterations) receive a sequence of tokens from the decoder.

Even though we are using the pre-trained ResNet-152 model as a base, we still found strong results with a BLEU score of 0.92 on our test set (as seen in Table 2). We attempted four major experiments, testing our variations on number of training epochs and the number of hidden layers in our decoder. We found consistently higher scores with a larger hidden layer count.

As expected, extended training caused the model to slowly over-fit the training data. We did not have the opportunity to mitigate this directly, but instead used a less-trained set of weights for more accurate results.

5.4 Discussion

The most surprising part of this project's success is how well a pre-trained image model can extract features from graphical interfaces, especially because they're not trained on them. However, we suspect that the pre-trained model is the source of most of the existing error, particularly around color-detection. What makes the system effective at the moment is likely the very simple DSL language. It would be interesting to experiment with a broader vocabulary (2+ orders of magnitude larger) and see if the BLEU scores hold up.

6 Conclusion/Future Work

To reiterate, this project looked to replicate and explore work done on *pix2code* in PyTorch. We approached the problem as an image-captioning problem, meaning we used a standard encoder/decoder

Model	**Training (BLEU Score)**	**Test (BLEU Score)**	**Training Set Size (#)**	**Dev Set Size (#)**	**Test Set Size (#)**
100 epochs, hidden_size = 512	0.95	0.92	1360	170	170
500 epochs, hidden_size = 512	0.99	0.90	1360	170	170
100 epochs, hidden_size = 256	0.85	0.76	1360	170	170
500 epochs, hidden_size = 256	0.93	0.84	1360	170	170

Table 2: Results for varying models.

pair of models to build our system. We used small experiments to affect our BLEU scores, which peaked at 0.92.

There is definitely room for more exploration — at this point, the system is more of a proof of concept to expand on. We wanted to create an end-to-end model which eliminates the Bootstrap-based DSL and pre-trained CNN, but lacked the time get it working. There is also more room to tweak hyper-parameters and experiment further.

7 Contributions

Andrew S. Lee (myself) was the sole contributor to this project, including: all write-ups, research, coding, and presentation.

See the code at its GitHub Repo.

References

[1] Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. *arXiv preprint arXiv:1705.07962*, 2017.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[5] Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. Machine learning-based prototyping of graphical user interfaces for mobile apps, 2018.

[6] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[7] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[8] Amanda Swearngin, Mira Dontcheva, Wilmot Li, Joel Brandt, Morgan Dixon, and Andrew J Ko. Rewire: Interface design assistance from examples. 2018.