Generating Webpages from Screenshots

Andrew S. Lee
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 takes screenshots as input and passes images into a ResNet-152-based CNN encoder, which generates features for a custom decoder RNN model. The project resulted in a peak BLEU score of 8.92 after a few hundred epochs.

Models
All of our features are gathered from a pre-trained ResNet-152 (size 1×1×254×8 per screenshot) model. While the model was not trained on GUI images[2], it does surprisingly well at extracting backgrounds, edges, colors, and text. This meant it was an easy and appropriate base to build our system on.

Data
Our data used pix2code’s generated screenshots based on a Bootstrap-based DSL vocabulary (18 words). It contains 1,250 pairs of 2480×1388 color images and their associated DSL code. We converted the image dimensions to 224×224 to use with ResNet-152.

Encoder Model
The encoder model is based on a pre-trained ResNet-152 model. We replace the final collection layer in order to collect a feature vector, which we then pass through a linear layer.

Decoder Model
The decoder model takes as inputs 1) the extracted features from the encoder model and 2) their target captions (DSL code put into a word embedding). It uses an LSTM, which we teach a language model based on the input features.

\[
L = \phi(W_1) + b_1 + \bar{W}_1h_1 + \bar{h}_1
\]

\[
\alpha_1 = \sigma(W_2 + h_1 + W_4\bar{h}_1 + \bar{h}_2)
\]

\[
\bar{h}_2 = \phi(W_3) + b_2 + \bar{W}_2\alpha_1 + \bar{h}_1
\]

\[
\beta_1 = \sigma(W_4 + \bar{h}_2 + W_5\alpha_1 + \alpha_2)
\]

\[
g_1 = \tanh(W_6 + h_1 + \bar{W}_6\alpha_1 + \beta_1)
\]

\[
\bar{h}_1 = \tanh(W_7 + \bar{h}_2 + W_8\alpha_1 + \bar{h}_1)
\]

Above: Equations for multi-layer LSTM RNN.

Results
We are using Bilingual Evaluation Understudy Scores (BLEU) to quantify our results, which is common for image-captioning models[2].

<table>
<thead>
<tr>
<th>Model</th>
<th>Training</th>
<th>Test</th>
<th>Train Set</th>
<th>Dev Set</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 epochs</td>
<td>hidden_size=512</td>
<td>8.95</td>
<td>8.92</td>
<td>1360</td>
<td>170</td>
</tr>
<tr>
<td>500 epochs</td>
<td>hidden_size=512</td>
<td>8.99</td>
<td>8.98</td>
<td>1360</td>
<td>170</td>
</tr>
<tr>
<td>500 epochs</td>
<td>hidden_size=256</td>
<td>8.85</td>
<td>8.76</td>
<td>1360</td>
<td>170</td>
</tr>
<tr>
<td>500 epochs</td>
<td>hidden_size=128</td>
<td>0.38</td>
<td>0.84</td>
<td>1360</td>
<td>170</td>
</tr>
</tbody>
</table>

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.

Future
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 to get it working. There is also more room to tweak hyper-parameters and experiment further.

References