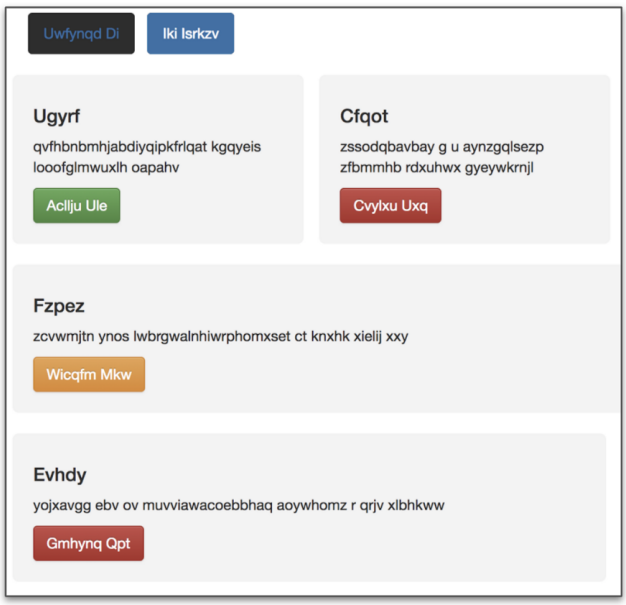
Ashwin Kumar Follow
Mar 20 · 8 min read

Automated front-end development using deep learning

SketchCode: Go from idea to HTML in 5 seconds



Ashwin Kumar was previously the co-founder of Sway Finance, a Y Combinator-backed startup that used machine learning to automate accounting. At Insight, he developed a model that allows users to create working HTML websites from hand-drawn wireframes, significantly accelerating the design process. He is now a Deep Learning Scientist at Mythic.

Apply now to join the next cohort of researchers and engineers building cutting-edge Applied AI products. *The deadline for the next Insight AI Fellowship (both Silicon Valley and New York) is March 26th.*

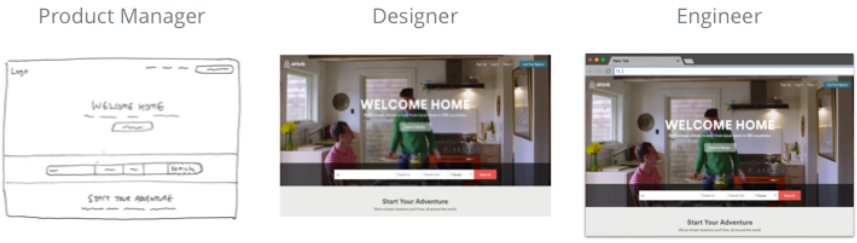
. . .

Creating intuitive and engaging experiences 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. Large corporations like Facebook have the bandwidth to dedicate entire teams to the design

process, which can take several weeks and involve multiple stakeholders; small businesses don't have these resources, and their user interfaces may suffer as a result.

My goal at Insight was to use modern deep learning algorithms to significantly streamline the design workflow and empower *any* business to quickly create and test webpages.

The design workflow today

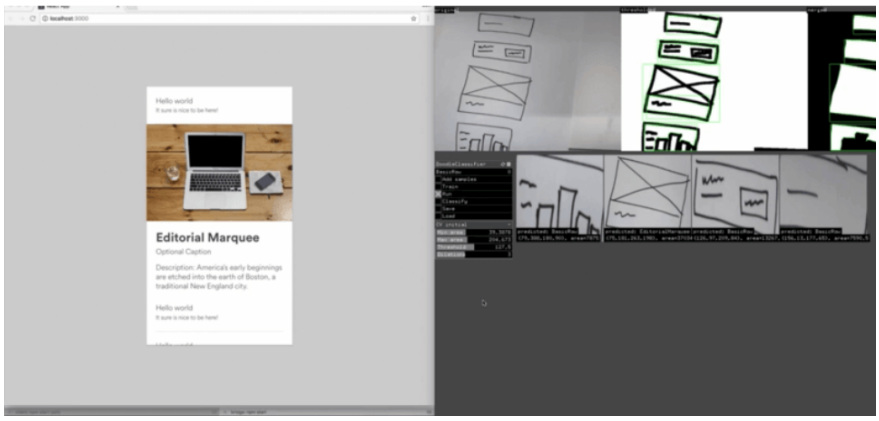


A design workflow goes through multiple stakeholders

A typical design workflow might look like the following:

- Product managers perform user research to generate a list of specifications
- Designers take those requirements and explore low-fidelity prototypes, eventually creating high-fidelity mockups
- Engineers implement those designs into code and finally ship the product to users

The length of the development cycle can quickly turn into a bottleneck, and companies like Airbnb have started to use machine learning to make this process more efficient.



Airbnb's demo of their internal AI tool to go from drawings to code

Though promising as an example of machine-assisted design, it's unclear how much of this model is being fully trained end-to-end, and how much relies on hand-crafted image features. There's no way to know for sure, because it's a closed-source solution proprietary to the company. I wanted to create an open-source version of **drawing-to-code** technology that's accessible to the wider community of developers and designers.

Ideally, my model would be able to take a simple hand-drawn prototype of a website design, and instantly generate a working HTML website from that image:

The SketchCode model takes drawn wireframes and generates HTML code

In fact, the example above is an actual website generated from my model on a test set image! You can check out the code on my Github page.

Drawing inspiration from image captioning

The problem I was solving falls under a broader umbrella of tasks known as program synthesis, the automated generation of working source code. Though much of program synthesis deals with code

generated from natural language specifications or execution traces, in my case I could leverage the fact that I had a source image (the hand-drawn wireframe) to start with.

There's a well-studied domain in machine learning called image captioning that seeks to learn models that tie together images and text, specifically to generate descriptions of the contents of a source image.

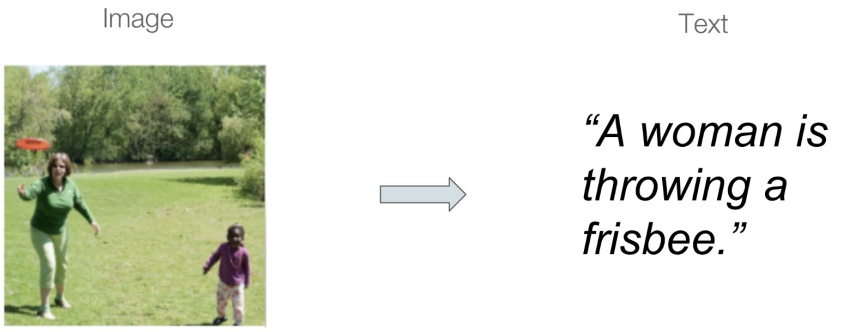


Image captioning models generate descriptions of source images

Drawing inspiration from a recent paper called pix2code and a related project by Emil Wallner that used this approach, I decided to reframe my task into one of image captioning, with the drawn website wireframe as the input image and its corresponding HTML code as its output text.

Getting the right dataset

Given the image captioning approach, my ideal training dataset would have been thousands of pairs of hand-drawn wireframe sketches and their HTML code equivalents. Unsurprisingly, I couldn't find that exact dataset, and I had to create my own data for the task.

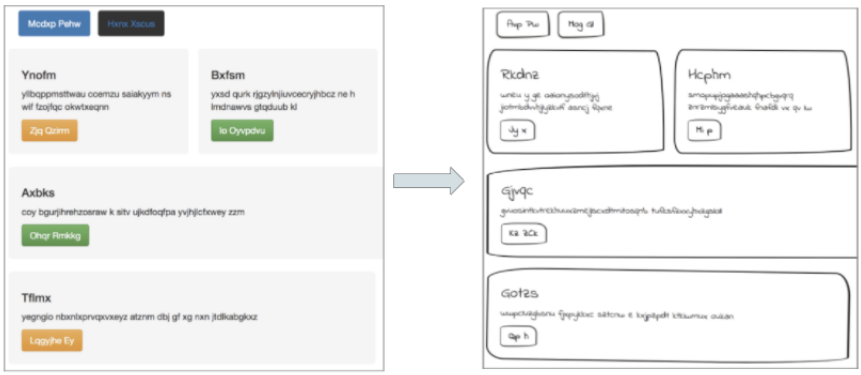
I started with an open-source dataset from the pix2code paper, which consists of 1,750 screenshots of synthetically generated websites and their relevant source code.

The pix2code dataset of generated website images and source code

This was a great dataset to start with, with a couple interesting points:

- Each generated website in the dataset consists of combinations of just a few simple Bootstrap elements such as buttons, text boxes, and divs. Though this means my model would be limited to these few elements as its 'vocabulary'—the elements it can choose from to generate websites—the approach should easily generalize to a larger vocabulary of elements
- The source code for each sample consists of tokens from a domain-specific-language (DSL) that the authors of the paper created for their task. Each token corresponds to a snippet of HTML and CSS, and a compiler is used to translate from the DSL to working HTML code

Making the images look hand-drawn



Turning colorful website images into hand-drawn versions

In order to modify the dataset for my own task, I needed to make the website images look like they were drawn by hand. I explored using

tools like OpenCV and the PIL library in python to make modifications to each image such as grayscale transformations and contour detection.

Ultimately, I decided to directly modify the CSS stylesheet of the original websites, performing a series of operations:

- Changed the border radius of elements on the page to curve the corners of buttons and divs
- Adjusted the thickness of borders to mimic drawn sketches, and added drop shadows
- Changed the font to one that looks like handwriting

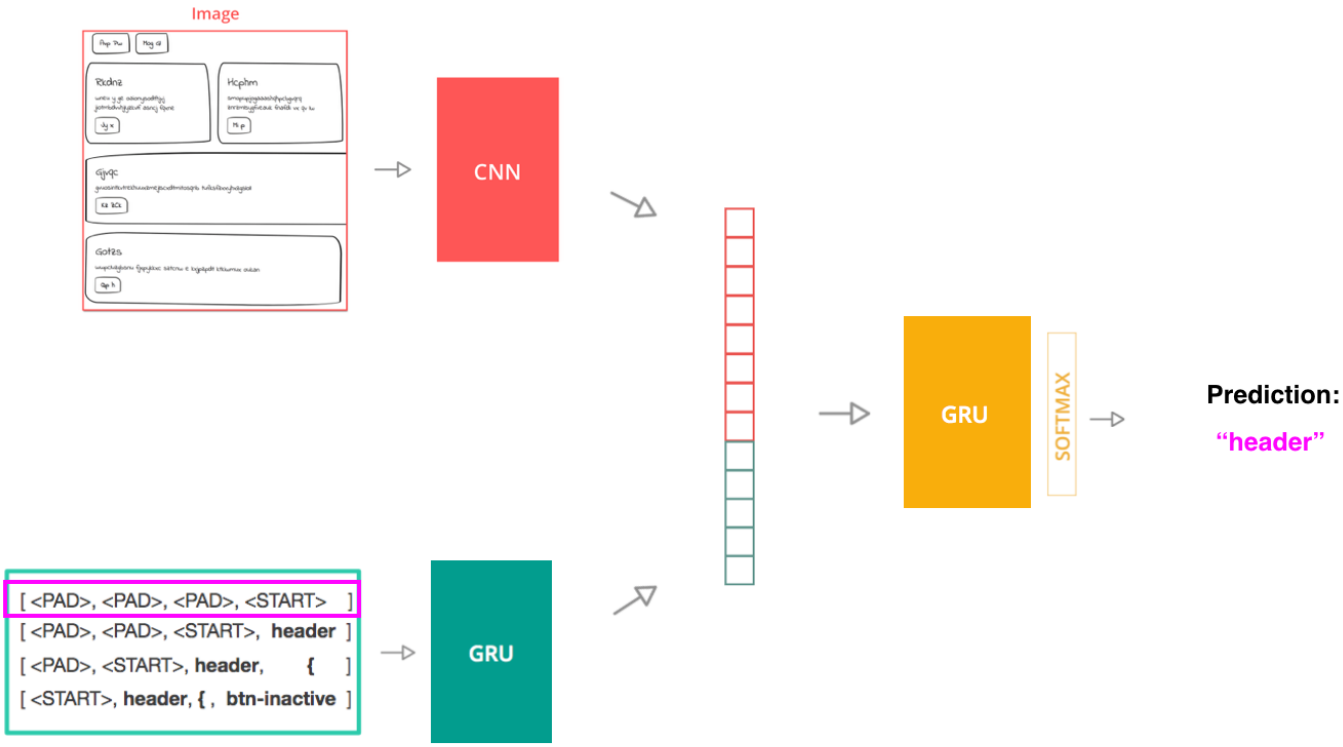
My final pipeline added one further step, which augmented these images by adding skews, shifts, and rotations to mimic the variability in actual drawn sketches.

Using an image captioning model architecture

Now that I had my data ready, I could finally feed it into the model!

I leveraged a model architecture used in image captioning that consists of three major parts:

- A computer vision model that uses a Convolutional Neural Network (CNN) to extract image features from the source images
- A language model consisting of a Gated Recurrent Unit (GRU) that encodes sequences of source code tokens
- A decoder model (also a GRU), which takes in the output from the previous two steps as its input, and predicts the next token in the sequence



Training the model using sequences of tokens as input

To train the model, I split the source code into sequences of tokens. A single input for the model is one of these sequences along with its source image, and its label is the next token in the document. The model uses the cross-entropy cost as its loss function, which compares the model's next token prediction with the actual next token.

At inference time when the model is tasked with generating code from scratch, the process is slightly different. The image is still processed through the CNN network, but the text process is seeded with just a starting sequence. At each step, the model's prediction for the next token in the sequence is appended to the current input sequence, and fed into the model as a new input sequence. This is repeated until the model predicts an <END> token or the process reaches a predefined limit to the number of tokens per document.

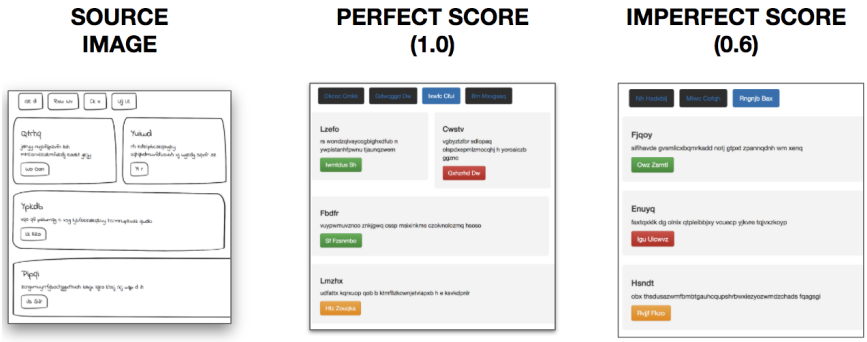
Once the set of predicted tokens is generated from the model, a compiler converts the DSL tokens into HTML, which can be rendered in any browser.

Evaluating the model with the BLEU score

I decided to use the BLEU score to evaluate the model. This is a common metric used in machine translation tasks, which seeks to measure how closely a machine-generated text resembles what a human would have generated, given the same input.

Essentially, the BLEU compares n-gram sequences of both the generated text and reference text to create a modified form of precision. It's very suitable for this project since it factors in the actual elements in the generated HTML, as well as where they are in relation to each other.

And the best part—I could actually *see* the BLEU scores through examining the generated websites!



Visualizing the BLEU score

A perfect BLEU score of 1.0 would have the right elements in the right locations given the source image, while a lower score would predict the wrong elements and/or put them in the wrong locations relative to each other. The final model was able to get a BLEU score of 0.76 on the evaluation set.

Bonus—Custom styling

An added bonus I realized was that since the model only generates the **skeleton** of the page (the tokens of the document), I could add in a custom CSS layer in the compilation process, and instantly have different styles of what the resulting website could look like.

One drawing => many styles generated simultaneously

Having the styling decoupled from the model generation process gives a couple big advantages to using the model:

- A front-end engineer who wants to incorporate the SketchCode model into their own company's product can use the model as-is and just change a single CSS file to comply with their company's style guide
- Scalability is built in—with a single source image the model outputs can instantly be compiled to 5, 10 or 50 different pre-defined styles, so users can visualize multiple versions of what their website might look like, and navigate around those websites in a browser

Conclusion and future directions

By leveraging research in image captioning, SketchCode is able to take hand-drawn website wireframes and convert them into working HTML websites in a few seconds.

The model has some limitations, which inform possible next steps:

- Since the model was trained on a vocabulary of just 16 elements, it can't predict tokens outside of what it's seen in the data. One next step could be to generate additional website examples using more elements, such as images, dropdown menus, and forms—Bootstrap components are a great place to start
- There's a lot of variation in actual production websites. A great way to create a training dataset more reflective of this variation would be to scrape actual websites, capturing their HTML/CSS code as well as screenshots of the site content
- Drawings also have a lot of variation that the CSS modification trick doesn't fully catch. A great way to generate more variation in

the hand-drawn sketch data could be to use a Generative Adversarial Network to create realistic-looking drawn website images

You can find the code used for this project on my GitHub page here, and I'm excited to see where the project goes next!

. . .

Want to learn applied Artificial Intelligence from top professionals in Silicon Valley or New York? Learn more about the Artificial Intelligence program.

Are you a company working in AI and would like to get involved in the Insight AI Fellows Program? *Feel free to get in touch.*

Thanks to Emmanuel Ameisen, Ben Regner, and Jeremy Karnowski.