Design2Struct: Generating Website Structures from Design Images using Neural Networks

Meine Matthias Velzel
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
m.m.velzel@student.utwente.nl

ABSTRACT

The task of translating visual design images into actual websites is a task usually done by human developers, this process can be slow, costly, and takes time away from implementing the actual functionality. In this paper, we address this problem by proposing a novel neural network architecture named *Design2Struct*. It makes use of Bahdanau Attention in an encoder-decoder structure to generate a sequence describing the website structure in a Domain Specific Language, which can then be compiled to code. The experimental evaluation shows that the proposed method outperforms the state-of-the-art methods by a large margin. Auxiliary, we identify that the existing benchmark dataset is oversimplified, and we propose a new benchmark dataset which is more realistic and one order of magnitude larger than the existing one.

1. INTRODUCTION

Developing a website is a process that goes through many phases, the first phase usually being designing the look of the website. This can be done by professional developers, but this is more often done by professional visual designers. The translation of the design to actual code, however, is a task that does still have to performed by developers, taking away time from implementing actual functionality and logic. In this paper, several contributions are proposed that aim to help computers learn to perform this task, allowing developers to spend their time more efficiently.

The first contribution is *Design2Struct*[1], a novel approach that uses neural networks to convert a *Graphical User Interface* (GUI) image to a structure describing the website, which can then be compiled to code. The approach is based on the model proposed by Beltramelli [2], which used Convolutional and Recurrent Neural Networks to generate a such structures. The novelty of *Design2Struct* is the introduction of *Bahdanau Attention* [1] to the previous work.

The second contribution is the release of a large *CommonCrawl*[2] based dataset, filtered and transformed to be used in the field of GUI to structure conversion. The dataset is publicly available[3] for use in future research.

The methodology of the research was setup in a way to answer three main research questions.

> **RQ1**: Can machine learning be used to convert GUI images to website structures?
>
> **RQ2**: What neural network architecture is most suitable for converting GUI images to website structures?
>
> **RQ3**: Is it possible to improve the performance of *pix2code* [2] with state-of-the-art methods?

The first and third questions are mainly related to the quality and performance of the work, while the second question is more related to the design of the work.

2. RELATED WORK

The generation of code or structure of web application from design images is a field of research not yet explored thoroughly.

The most important contribution to this field is *pix2code* by Beltramelli [2], who proposed a novel approach by applying techniques from the field of Image Captioning and Natural Language Processing (NLP) to code generation by generating a *Domain Specific Language* (DSL). This DSL could then be further compiled to functioning code. Their proposed method is based on *Convolutional Neural Networks* (CNNs) and *Recurrent Neural Networks* (RNNs).

2.1 Image Captioning

A field very similar to the field explored by *Design2Struct* and *pix2code* is the field of image captioning. Image captioning also involves converting images to language, albeit natural language instead of DSLs.

Traditional and widely used image captioning methods usually involve a CNN followed by an RNN, a well known example being "NIC" by Vinyals et al. [17]. Other more state-of-the-art methods involve attention mechanisms, which have been shown to be very powerful at highlighting important parts of images. Some methods with advanced attention mechanisms even forego the use of RNNs. The current highest performing model in the "Image Captioning Challenge" on the *MSCOCO* [12] dataset by Pan et al. [13] makes abundant use of attention mechanisms.

2.2 Natural Language Processing

A field that very closely tied to image captioning, and therefore also very relevant to this research, is the field of Natural Language Processing (NLP).

[1]https://github.com/mvelzel/Design2Struct
[2]https://commoncrawl.org
[3]https://www.kaggle.com/meinevelzel/webcrawl-bootstrap-compiled

33rd Twente Student Conference on IT July. 3rd, 2020, Enschede, The Netherlands.

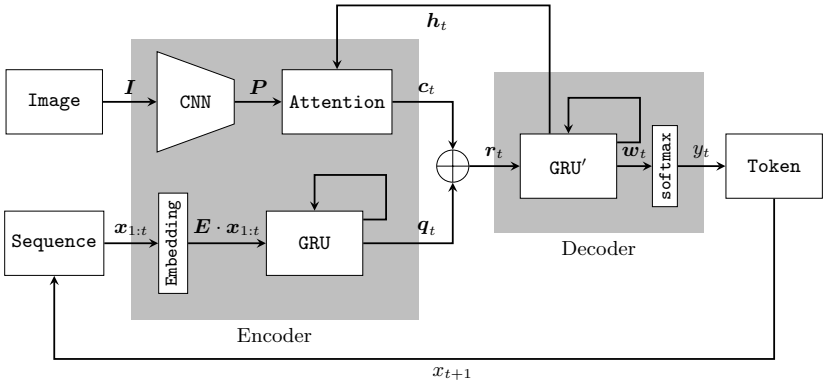


Figure 1: Overview of the *Design2Struct* architecture. The GUI image is encoded by the CNN based image encoder. Its features are then highlighted by the subsequent attention mechanism, conditioned on the hidden state of the decoder after the previous time step. The context sequence (a sequence of embedded tokens corresponding to DSL) is encoded by the language encoder, consisting of a GRU layer. The two resulting vectors are then concatenated and fed into the decoder, consisting of another GRU layer. Finally, *softmax* is used to sample one token at a time. In training it is compared to the ground-truth token, in sampling it is appended back to the sequence until the <*END*> token is predicted.

Traditional NLP methods heavily involve the use of RNNs. An example used for machine translation is by Cho et al. [5] who first proposed the encoder-decoder network model, using an RNN to encode the input sentence, followed by an RNN to decode it to another language. In order to improve such networks, attention mechanisms have widely been adopted. One popular example is by Bahdanau et al. [1], who proposed a novel attention mechanism to better highlight relevant words from the input sentence when it generates certain words.

Attention mechanisms have been shown to be very powerful, to the point that most state-of-the art Natural Language models now make exclusive use of these attention mechanisms. These models, dubbed "transformer" models, were first introduced by Vaswani et al. [16], and were further used by *OpenAI* in their *GPT-2* [15] and *GPT-3* [3] models.

3. DESIGN2STRUCT

In this section, we present our novel proposed model, *Design2Struct*. *Design2Struct* consists of an aforementioned encoder-decoder network. A CNN is used as the encoder, and an RNN as the decoder. Such a model was first applied to GUI structure generation by *pix2code* [2]. Such an encoder-decoder network was improved by Xu et al. [18] with the use of the attention mechanism proposed by Bahdanau et al. [1]. *Design2Struct* combines both approaches to end up with a novel architecture in the field of GUI structure generation.

3.1 Image Encoder

CNNs are currently the method of choice for many vision problems because of their powerful ability to identify important features of the images they are trained on. A CNN is used in the model to encode an input image as a set of F vectors $\boldsymbol{p}_i, i \in \{0...F\}$ of size U, or the matrix $\boldsymbol{P}_{F \times U}$, corresponding to the features extracted at different image locations. These are then fed further in the model, as is shown in Figure 1.

The input images are resized to 299×299 pixels (aspect-ratio not preserved) with the pixel values normalized before being fed into the CNN. To encode each image as fixed-length vectors, 3×3 receptive fields convolved with stride 1 were used. These operations are applied once before dimensionality reduction with the same fields but with stride 2. The width of the first convolutional layer is 16, followed by a layer of width 32, then width 64, and finally width 128.

3.2 Language Encoder

In order to describe the structure of simple websites, a DSL was designed based on the DSL used by *pix2code* [2]. This DSL is illustrated in Figure 2. The current work is only interested in the layout and elements of the design; thus the textual value of the elements is ignored. The size of the vocabulary and its specific elements can be found in Appendix A.

The tokens in the vocabulary are encoded using an Embedding layer and can be further encoded by an RNN encoder, as is shown in Figure 1. *Design2Struct* encodes the tokens with both an Embedding and an RNN layer, but models without this RNN layer were also tested. Results will be discussed in Section 4.

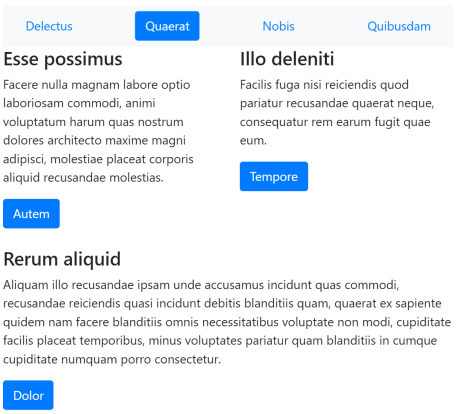
In the DSL an element is declared with an opening token; if several elements are contained within a block, a closing token is also needed for the compiler. In the case where several children elements are contained within a parent element, the model has to keep track of long-term dependencies in order to close an opened block. Traditional RNN architectures suffer from vanishing or exploding gradients when dealing with such long-term dependencies, therefore Hochreiter and Schmidhuber [9] proposed the *Long Short Term Memory* (LSTM) architecture to address this problem. While *pix2code* [2] opted for this architecture, *Design2Struct* makes use of the *Gated Recurrent Unit* (GRU) [5]. This is an architecture based LSTMs, but uses less gates and is therefore less computationally expensive, while its performance stays roughly the same [6].

The GRU encoding layer is implemented as a single GRU layer with 128 cells.

3.3 Attention Mechanism

An attention mechanism is a mechanism that gives weights to features of an image, depending on the token parsed at a certain time step. This allows a model to learn what parts of a context image are more or less relevant, depending on the parsed token.

The attention mechanism used in *Design2Struct* was first introduced by Bahdanau et al. [1] for use in machine trans-



(a) Bootstrap GUI screenshot

```
Body {
    Header {
        Link
        Button
        Link
        Link
    }
    Row {
        Column {
            Subtitle
            Paragraph
            Button
        }
        Column {
            Subtitle
            Paragraph
            Button
        }
    }
    Row {
        Column {
            Subtitle
            Paragraph
            Button
        }
    }
}
```

(b) Code describing the GUI written in the DSL

Figure 2: An example of a simple Bootstrap based website written in the *pix2code* based DSL

lation. The mechanism was then applied to the field of image captioning by Xu et al. [18]. This mechanism generates a context vector $\boldsymbol{c}_t$, which is a dynamic representation of the relevant parts of a context image at time t. The context vector is computed from the feature matrix $\boldsymbol{P}_{F\times U}$ resulting from parsing a context image through the CNN encoder.

For each feature extracted at different image locations, a positive weight a_i is generated, which can be interpreted as the probability that location i is the right location to pay attention to. The weights $a_i, i \in \{0...F\}$ for the feature matrix $\boldsymbol{P}_{F\times U}$ are computed by an *attention model* f_{att} which is implemented as a *Multilayer Perceptron* (MLP) conditioned on the decoder RNN's previous hidden state $\boldsymbol{h}_{t-1}$. The architecture of the MLP can be found in Figure 7.

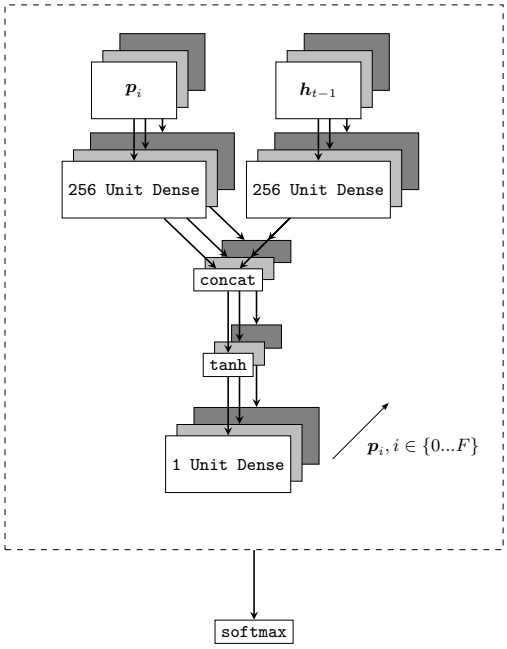


Figure 3: Diagram of the f_{att} Multilayer Perceptron used in the attention mechanism.

The weights are then computed as follows:

$$a_i = f_{att}(\boldsymbol{P}_{F\times U}, \boldsymbol{h}_{t-1}), i \in \{0...F\} \tag{1}$$

Once the weights (which after *softmax* sum to one) are computed, the context vector $\boldsymbol{c}_t$ is computed as a weighted sum by:

$$\boldsymbol{c}_t = \sum_{i=1}^{F} a_i \boldsymbol{p}_i \tag{2}$$

3.4 Decoder

The model is trained in a supervised manner with an image $\boldsymbol{I}$ and a context sequence $\boldsymbol{x}$ of T token embeddings $\boldsymbol{E} \cdot x_t, t \in \{0...T-1\}$ as inputs. The output vectors of the image and language encoders are concatenated and then fed into the decoder. It then decodes this information as an output token, learning the relationships between the context image and sequence.

The decoder is implemented as a single GRU layer with 256 cells, followed by a *softmax* layer the size of the vocabulary, which is used to sample single tokens at a time.

The entire architecture can be expressed mathematically as follows:

$$\boldsymbol{P}_{F\times U} = CNN(\boldsymbol{I}) \tag{3}$$
$$\boldsymbol{q}_t = GRU(\boldsymbol{E} \cdot \boldsymbol{x}_{1:t}) \tag{4}$$
$$a_i = f_{att}(\boldsymbol{P}_{F\times U}, \boldsymbol{h}_{t-1}), i \in \{0...F\} \tag{5}$$
$$\boldsymbol{c}_t = \sum_{i=1}^{F} a_i \boldsymbol{p}_i \tag{6}$$
$$\boldsymbol{r}_t = (\boldsymbol{q}_t, \boldsymbol{c}_t) \tag{7}$$
$$\boldsymbol{w}_t, \boldsymbol{h}_t = GRU'(\boldsymbol{r}_t) \tag{8}$$
$$y_t = softmax(\boldsymbol{w}_t) \tag{9}$$
$$x_{t+1} = y_t \tag{10}$$

3.5 Training

The length T of the context sequences used for training is important to model long-term dependencies. A length T of 1 was used in the model by Xu et al. [18], this was possible because of the powerful attention mechanism, combined with the fact that image descriptions do not hold as many long-term dependencies. A length T of 1 and 64 were both tested with experiments which will be discussed in Section 4. *Design2Struct* uses a length T of 64, meaning a sliding window of length 64, during training.

While the context sequence of tokens used for training is

updated for each new token by sliding the window, the same image $\boldsymbol{I}$ is used for each window in the same sequence.

There are two extra special tokens, $<START>$ and $<END>$, used to respectively prefix and suffix the token sequences in order to indicate their start and end.

Training is performed by computing the partial derivatives of the loss function with respect to the whole network's weights, calculated using backpropagation to minimize the loss function. The multiclass log loss used for training the network is as follows:

$$L(x_{t+1}, y_t) = -\sum_{t=1}^{N} x_{t+1} \log(y_t) \tag{11}$$

With x_{t+1}, y_t, and N, being the predicted token, the expected token, and the vocabulary size, respectively. The model is optimized end-to-end so loss L is minimized with respect to all model parameters, this includes the encoders, the attention mechanism, and the decoder. The training was done with the *Adam* [10] optimizer with the learning rate set to $1e-3$.

To prevent overfitting, dropout regularization was used both in the CNN and RNN networks. In the CNN a 20% dropout layer was used after each dimension reducing layer. The GRU layers also used a dropout of 20%, which was only applied to the non-recurrent connections. The model was trained with batches of 8 image-sequence pairs.

The full process of a single training step for *Design2Struct* is described by Algorithm 1.

4. EXPERIMENTS AND RESULTS

In this section first the different datasets used in the experiments will be discussed. Followed by the methodology shaped around the research questions. Finally the actual results of all experiments will be presented and discussed.

4.1 Datasets

All datasets used were made to conform to a uniform style, which was the default style of the CSS framework Bootstrap 4[4]. GUI images were generated by compiling the DSL to a simple HTML and CSS page with the default Bootstrap 4 style. The algorithm for this compilation is the same algorithm used by *pix2code* [2], but with a different DSL-Class to HTML-Node mapping. No real text was used in the compiled websites and images, instead random words and paragraphs generated with *Lorem Ipsum* were used.

4.1.1 pix2code based

The dataset used for most experiments is the same dataset used by *pix2code* [2], with a couple transformations applied. First, the DSLs used by the *pix2code* [2] dataset were translated to the DSL defined by this paper using the simple mapping described by Appendix B. Then, screenshots were generated with the new DSLs by using the previously mentioned algorithm to create the new image-sequence pairs. The details of the dataset can be found in Table 4.

4.1.2 CommonCrawl based

The newly created dataset was made by first filtering through the large, publicly available, *CommonCrawl* dataset. After filtering, the data was transformed by converting the source HTML to the *Design2Struct* DSL by using the algorithm described in Appendix C.

The dataset was filtered based on a couple criteria:

1. The website was listed as being in English.
2. The website contained a reference to either `bootstrap.css` or `bootstrap.min.css` in its `<head>`.
3. The DSL resulting from converting the source HTML was a maximum length of 512.

From the generated DSLs two datasets were created. The first dataset contains the original GUI screenshots paired with the generated DSLs. The second contains new screenshots, generated by compiling the DSLs, paired with the generated DSLs. The details of the resulting datasets are both found under "CommonCrawl" in Table 4.

Algorithm 1: *Design2Struct* Single Training Step

Input: Maximum sequence length M
Maximum context length T
Optimizer function $f_{optimizer}$
Model weights and biases $\boldsymbol{W}$
Input image $\boldsymbol{I}$
Ground-truth sequence $\boldsymbol{y}_{1:M}$

Output: New model weights and biases θ_{new}

$Loss = 0$
$x_1 =< START >$
Set the context sequence beginning indicator $s = 1$
Pad the context sequence $\boldsymbol{x}_{s:1}$ to length T
Initialize $\boldsymbol{h}_0$ to all zeros
`/* Calculate the image feature matrix. */`
$\boldsymbol{P}_{F\times U} = CNN(\boldsymbol{I})$
for $t \leftarrow 1$ **to** M **do**
  `/* Use the rest of Design2Struct to predict the next token. */`
  $\boldsymbol{q}_t = GRU(\boldsymbol{E} \cdot \boldsymbol{x}_{s:t})$
  $a_i = f_{att}(\boldsymbol{P}_{F\times U}, \boldsymbol{h}_{t-1}), i \in \{0...F\}$
  $\boldsymbol{c}_t = \sum_{i=1}^{F} a_i \boldsymbol{p}_i$
  $\boldsymbol{r}_t = (\boldsymbol{q}_t, \boldsymbol{c}_t)$
  $\boldsymbol{w}_t, \boldsymbol{h}_t = GRU'(\boldsymbol{r}_t)$
  $x_{t+1} = softmax(\boldsymbol{w}_t)$
  `/* Calculate the loss and update the context sequence for` $t+1$. `*/`
  $Loss = Loss + L(x_{t+1}, y_t)$
  $\boldsymbol{x}_{s:t+1} = (\boldsymbol{x}_{s:t}, y_t)$
  $s = \max(1, t - T + 1)$
  Pad sequence $\boldsymbol{x}_{s:t+1}$ to length T
`/* Calculate the gradients and update the weights and biases accordingly. */`
Calculate gradients $\nabla_{\boldsymbol{W}} Loss$
$\boldsymbol{W}_{new} = f_{optimizer}(\nabla_{\boldsymbol{W}} Loss, \boldsymbol{W})$
Return $\boldsymbol{W}_{new}$

4.2 Methodology

The experiments were designed to properly provide answers to the three research questions.

To answer the first and the third research questions, proper evaluation of results is needed. For machine learning to be usable in converting GUI images to website structures, the model needs to achieve usable results. For the model to be a constructive contribution to the field, it must outperform past contributions like *pix2code* [2].

To answer the second research question, many architectures were tested and compared to determine the most suitable architecture.

[4]https://getbootstrap.com

Table 1: Model Results after 20 Epochs. The naming scheme of the unnamed models can be interpreted as follows: {Encoder with or without added RNN}_{Decoder RNN}_{Sequence to word or word to word}_{Optional *Bahdanau Attention* [1] following CNN}.

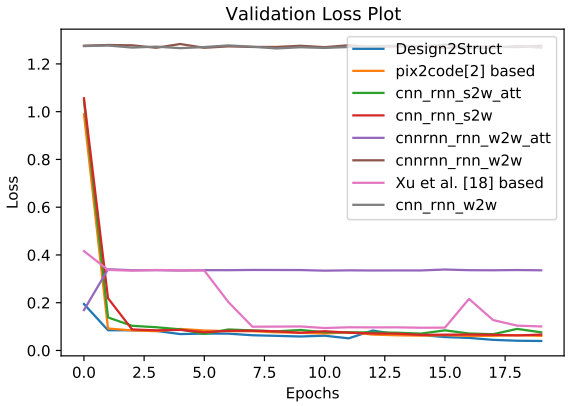
Model	Loss	Val. Loss	BLEU	ROUGE-1	ROUGE-2	ROUGE-L
Design2Struct	0.0649	0.0617	0.8286	0.8600	0.8318	0.9928
pix2code [2] based	0.0642	0.0613	**0.8433**	**0.8866**	**0.8628**	0.9962
cnn_rnn_s2w_att	0.0660	0.0617	0.8420	0.8782	0.8603	**0.9986**
cnn_rnn_s2w	0.0619	0.0708	0.8400	0.8724	0.8454	0.9951
cnnrnn_rnn_w2w_att	0.3403	0.3357	0.4851	0.6452	0.6078	0.8575
cnnrnn_rnn_w2w	1.2797	1.2760	0.0616	0.6070	0.1686	0.7207
Xu et al. [18] based	0.0970	0.0936	0.6400	0.7101	0.6791	0.9868
cnn_rnn_w2w	1.279	1.268	0.0653	0.6173	0.1729	0.7175

Table 2: Model Results after 50 Epochs. The naming scheme of the unnamed models can be interpreted as follows: {Encoder with or without added RNN}_{Decoder RNN}_{Sequence to word or word to word}_{Optional *Bahdanau Attention* [1] following CNN}.

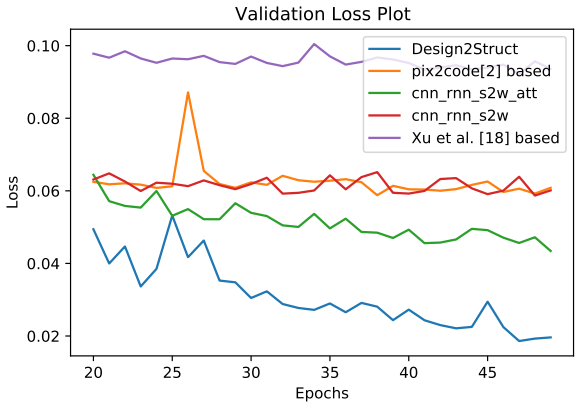
Model	Loss	Val. Loss	BLEU	ROUGE-1	ROUGE-2	ROUGE-L
Design2Struct	0.0207	0.0196	**0.9534**	**0.9737**	**0.9830**	**1.0000**
pix2code [2] based	0.0615	0.0608	0.8542	0.8939	0.8707	0.9936
cnn_rnn_s2w_att	0.0456	0.0434	0.8679	0.8712	0.8621	0.9988
cnn_rnn_s2w	0.0602	0.0601	0.8508	0.8913	0.8657	0.9937
Xu et al. [18] based	0.0955	0.0936	0.7115	0.7534	0.7214	0.9806

Table 3: Model results after 11 Epochs on the *CommonCrawl* based dataset.

Model	Loss	Val. Loss	BLEU	ROUGE-1	ROUGE-2	ROUGE-L
Design2Struct	0.3077	0.3951	0.3244	0.5128	0.3965	0.7132

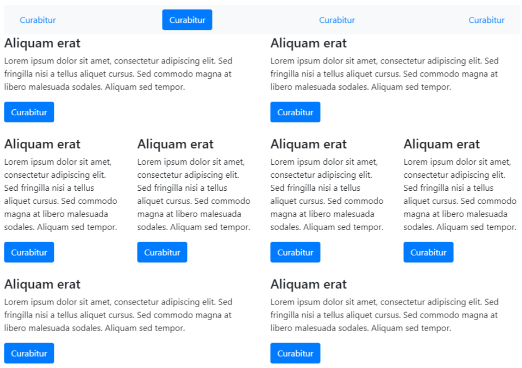


(a) Validation losses for preliminary experiments over 20 epochs.

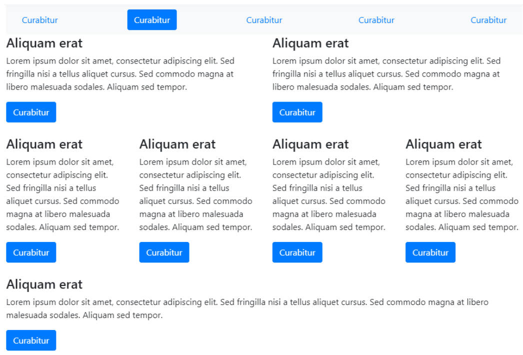


(b) Validation losses for in-depth experiments over the next 30 epochs.

Figure 4: Validation loss plots from the experiments ran on the *pix2code* [2] based dataset.



(a) Groundtruth GUI screenshot



(b) GUI screenshot predicted with *Design2Struct*. The only discrepancy is the lack of a second Subtitle, Paragraph, Button combination in the last row.

Figure 5: Experiment samples from the *pix2code* [2] based dataset

Table 4: Dataset sizes.

Dataset	Total Size	Instances		
		Training	Validation	Test
pix2code	212 MB	1225	175	350
CommonCrawl	5.56 GB	10995	1570	3143

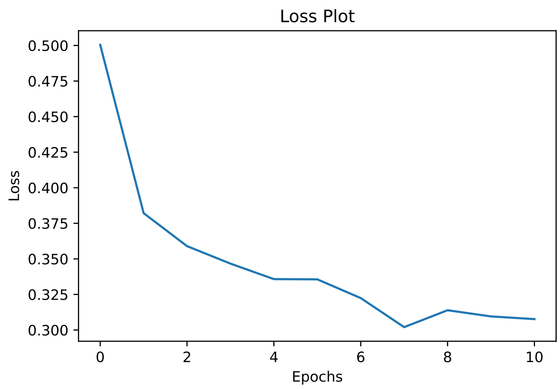


Figure 6: Losses for the larger dataset experiments over 11 epochs.

4.2.1 Architecture Design

To determine the most suitable neural network architecture, various different architecture varieties were combined and tested. A combination of the following architecture designs were tested:

- A Language Encoder with or without an added RNN layer over the initial embedding layer.
- A word-to-word or a sequence-to-word method, in other words, a context sequence length T of 1 or 64.
- An Image Encoder with or without a *Bahdanau Attention* [1] mechanism following the CNN.

All combinations of these varieties give a total of $2^3 = 8$ models to be tested and compared.

4.2.2 Evaluation

All models were tested with metrics used commonly in the fields of image captioning and NLP. The exact metrics used were BLEU [14] and ROUGE [11]. For all ROUGE metrics the F_1 score was used.

Both the BLEU [14] and ROUGE [11] metrics evaluate predicted sequences by comparing them against several reference sequences. They compare sequences by using n-gram overlap, n-grams being all subsequences of length n. Both metrics have a minimum score of 0 and a maximum of 1. BLEU [14] uses the precision of 4-gram overlap between the predicted sequence and the reference sequences. ROUGE-1 and ROUGE-2 [11] use 1-gram and 2-gram overlap, respectively. They use the F_1 score instead of precision, with the F_1 score being the harmonic mean of precision and recall.

These metrics are sufficiently representative of the model's relative performance, though one has to keep in mind that these metrics were designed for natural languages and not DSLs. For better evaluation, one could look at metrics for comparing graph structures, as the DSLs can be interpreted as a tree. This was, however, out of the scope of this research.

4.3 Results

4.3.1 Preliminary Experiments

For the first round of experiments, all 8 models were run for 20 epochs on the *pix2code* [2] based dataset. The validation loss plots can be found in Figure 4a and the evaluation results can be found in Table 1.

The results show very similar results among the models with a T of 64, whereas the models with a T of 1 generally performed much worse. It can be seen, however, that the model based on Xu et al. [18] performed much better than the others in that category. In the models with a T of 1 it very clearly shows that an attention mechanism is needed to capture long-term dependencies, whereas with the models with a T of 64 show no substantial difference, with *pix2code* barely ending on top.

4.3.2 In-depth Experiments

The second round of experiments was only done with the 5 most promising models due to time and computing power constraints. Each model this round was ran for 50 epochs on the *pix2code* [2] based dataset. The validation loss plots can be found in Figure 4b and the evaluation results can be found in Table 2.

The results clearly show that the *Design2Struct* architecture outperforms all other architectures by quite a large margin. The model based on Xu et al. [18] hits a clear

plateau, as can be seen in Figure 4b, most likely due to its lack of long-term dependencies. The models without attention also hit a clear plateau as shown by Figure 4b.

However, *Design2Struct* and the other model with attention did not hit such a plateau, and were therefore able to learn much more and perform better. Since loss curves have not hit a plateau yet, and the models have not yet started overfitting, it is entirely possible that they could perform even better if given more time. Only once the validation loss starts diverging from the training loss, the model will have started overfitting.

4.3.3 Larger Dataset Experiments

The final round of experiments was done only with the *Design2Struct* architecture on the larger *CommonCrawl* dataset. This was also due to time and computing constraints. The model was trained for roughly 70 hours and it reached 11 epochs. The evaluation results can be found in Table 3.

The results show that the performance of *Design2Struct* on a larger, more complex, dataset is substantially less, given less training time. This is mainly due to the large difference in relative complexity between the *pix2code* [2] and *CommonCrawl* based datasets. Evidence of this can be seen in the *pix2code* [2] to *Design2Struct* DSL mapping in Appendix B. The resulting classes used after the mapping are far fewer than the the classes used in the *CommonCrawl* based dataset, all of which can be found in Appendix A. This could be fixed in future research by creating a more one-to-one DSL mapping between *pix2code* [2] and *Design2Struct*.

It can also be seen, however, that the *Design2Struct* model can definitely still improve on this dataset given more time. This is because the loss plot in Figure 6 clearly shows a continuing downwards trend, with no sign of a plateau. Moreover, when comparing the training loss to the validation loss in Table 3, it can be seen that the model has not yet started overfitting.

5. CONCLUSION

In this paper *Design2Struct* was presented, an improvement over existing methods by introducing *Bahdanau Attention* [1] to the field GUI structure generation. The work demonstrates an improvement over previous methods, however the results could be even further improved with prolonged training over an improved *CommonCrawl* based dataset.

The quality of the results shows that it is clearly possible to convert GUI images to website structures, at least for simple, well-defined cases.

Experiments show that out of all tested architectures, *Design2Struct* performed the best, and is thus the most suitable neural network architecture out of the others.

The results of the experiments in this research cannot directly be compared with the results from the *pix2code* [2] paper, as the DSLs were different and the dataset was transformed. The architecture proposed by Beltramelli [2], however, was faithfully implemented and compared with the same DSL and datasets, and results show that *Design2Struct* outperforms *pix2code* [2] on all used metrics.

6. FUTURE WORK

The architecture could possibly further be improved with the use of more recent state-of-the art techniques in the fields of image captioning and NLP.

Generative Adversarial Networks (GANs) [8] have been shown to be very powerful at generating images and sequences. Use of such networks already has precedence in the field of image captioning [7, 4]. Applying such techniques to the field of structure generation is a not well researched as of yet. GANs could be used on their own, or as an improvement to the current *Design2Struct* encoder-decoder architecture, as was suggested by Chen et al. [4].

As mentioned in the state-of-the art, transformer models have now largely become the standard in NLP and have replaced RNN based architectures. Such models could replace the RNN structure that *Design2Struct* currently uses for a possible improvement on more complex use-cases. Application of transformer models to the field of GUI structure generation is also a research area not yet explored.

References

[1] Dzmitry Bahdanau et al. "End-to-end attention-based large vocabulary speech recognition". In: *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2016, pp. 4945–4949.

[2] Tony Beltramelli. "pix2code: Generating Code from a Graphical User Interface Screenshot". In: *arXiv preprint arXiv:1705.07962* (2017).

[3] Tom B Brown et al. "Language models are few-shot learners". In: *arXiv preprint arXiv:2005.14165* (2020).

[4] Chen Chen et al. "Improving image captioning with conditional generative adversarial nets". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 8142–8150.

[5] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[6] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[7] Bo Dai et al. "Towards diverse and natural image descriptions via a conditional gan". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2970–2979.

[8] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[10] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv: 1412.6980* (2014).

[11] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://www.aclweb.org/anthology/W04-1013.

[12] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[13] Yingwei Pan et al. "X-Linear Attention Networks for Image Captioning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

[14] Kishore Papineni et al. “BLEU: a Method for Automatic Evaluation of Machine Translation”. In: (Oct. 2002). DOI: 10.3115/1073083.1073135.

[15] Alec Radford et al. “Language models are unsupervised multitask learners”. In: *OpenAI Blog* 1.8 (2019), p. 9.

[16] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[17] Oriol Vinyals et al. “Show and tell: A neural image caption generator”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3156–3164.

[18] Kelvin Xu et al. “Show, attend and tell: Neural image caption generation with visual attention”. In: *International conference on machine learning*. 2015, pp. 2048–2057.

APPENDIX

A. DSL CLASSES

The different classes that the DSL can use to define the structure of a visual design image. They are split into layout classes, basic elements, and form elements. Not all of these classes are made use of in the *pix2code* [2] dataset, but all are used in the *CommonCrawl* dataset.

A.1 Layout Classes

Body Root of the visual part of the page, also always the root of the tree.

Block A section of information related to each other, can often be an image with a related paragraph and maybe some links or buttons.

Container A layout class that contains page general page content.

Row A layout class that defines rows in a page structure.

Column A layout class that defines columns in rows in a page structure.

Header A section of a page usually at the top that could contain a title and usually several navigation links.

Footer A section of a page usually at the bottom that usually contains some links to other related websites and possibly contact information.

A.2 Basic Elements

Paragraph A simple block of text.

Image An image.

Button A button, usually with a square shape that looks more defined than a link.

Subtitle A small heading.

Title A large heading.

Link A simple hyperlink.

A.3 Form Elements

TextBox A box that can be used for any sort of text input.

CheckBox A box or item with two states, usually checked or unchecked.

RadioBox A box or item usually part of a list of other radio boxes where only one box in that list can be selected.

Range An element, usually a slider of some sort, that can be used to define a range.

B. MAPPING FROM PIX2CODE DSL TO DESIGN2STRUCT DSL

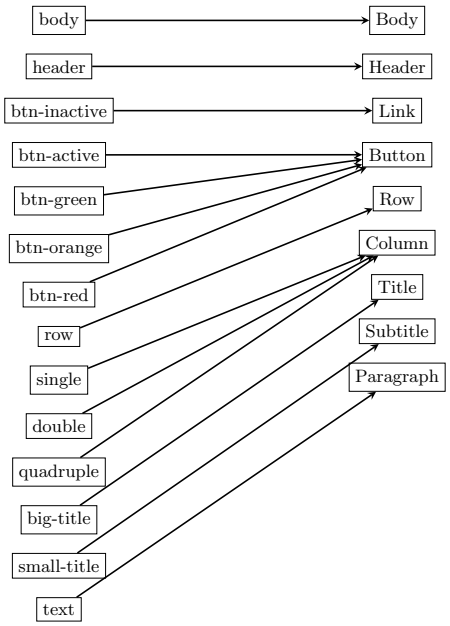


Figure 7: A mapping from classes in the *pix2code* [2] Bootstrap DSL to the *Design2Struct* DSL.

C. ALGORITHM FOR HTML SOURCE TO DSL CONVERSION

Algorithm 2: HTML to DSL conversion

Input: HTML tree root R
Mapping M from HTML Node to Class
Output: Formatted DSL String

Function `ParseNode`(*Node N, Parent Node P*)**:**
  Get Class C from M
  if *No Class C found for N in M* **then**
    Children Parent $CP = P$
  else
    Children Parent $CP = C$
  if *C is the same as P* **then**
    C is No Class
  Result String $RS =$ “ ”
  for *Child Node CH of N* **do**
    $RS = RS +$ ParseNode(CH, CP) + “ ”;
  if *C is No Class* **then**
    Return RS
  if *RS is “ ”* **then**
    Return $C +$ “ { ” $+ RS +$ “}”
  Return C
Return ParseNode(R)