Automatic Generation of Code from Screenshots

Namrata R
PES1201700921
PES University

Sarang R
PES1201700972
PES University

Pragnya S
PES1201701342
PES University

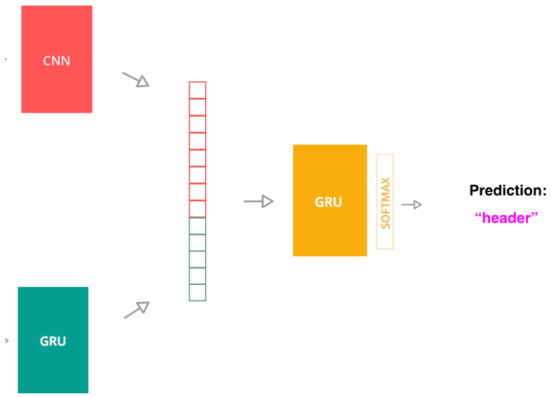
I. Project Abstract

This project aims on cutting down development time of web UI design by generating responsive web layouts straight from images of webpages or even hand-drawn sketches of webpages that is, to generate HTML code from images. This problem can be modelled as an image-captioning problem where the output language consists of a series of tokens which can be converted into HTML code with the help of a compiler.

II. Project Scope

Creating intuitive and engaging experiences for users is a critical goal for all web designers and developers.This project enables one to build webpages in a short span of time. Modern deep learning algorithms can be used to significantly streamline the design workflow and help quickly create and test webpages. On developing an accurate model, this concept can easily be extended to a general problem of UI development. This helps a developer focus more on the functionality, rather than the layout.

III. Model Architecture



1) Image Model

Layer (type)	Output Shape	Param #
conv2d_19 (Conv2D)	(None, 224, 224, 32)	896
conv2d_20 (Conv2D)	(None, 224, 224, 32)	9248
max_pooling2d_10 (MaxPooling	(None, 112, 112, 32)	0
dropout_16 (Dropout)	(None, 112, 112, 32)	0
conv2d_21 (Conv2D)	(None, 112, 112, 64)	18496
conv2d_22 (Conv2D)	(None, 112, 112, 64)	36928
max_pooling2d_11 (MaxPooling	(None, 56, 56, 64)	0
dropout_17 (Dropout)	(None, 56, 56, 64)	0
conv2d_23 (Conv2D)	(None, 56, 56, 128)	73856
conv2d_24 (Conv2D)	(None, 56, 56, 128)	147584
max_pooling2d_12 (MaxPooling	(None, 28, 28, 128)	0
dropout_18 (Dropout)	(None, 28, 28, 128)	0
flatten_4 (Flatten)	(None, 100352)	0
dense_10 (Dense)	(None, 1024)	102761472
dropout_19 (Dropout)	(None, 1024)	0
dense_11 (Dense)	(None, 1024)	1049600
dropout_20 (Dropout)	(None, 1024)	0
batch_normalization_4 (Batch	(None, 1024)	4096
repeat_vector_4 (RepeatVecto	(None, None , 1024)	0

A Convolutional Neural Network has been used as an encoder in order to learn and identify features of the given screenshots.The model taken in images of shape(224,224,3) and has multiple convolutional layers an 2 fully connected layers.Finally the output it is repeated input sequence number of times

2) Language Model

Layer (type)	Output Shape
embedding_5 (Embedding)	(None, None , 50)
gru_9 (GRU)	(None, None , 256)
gru_10 (GRU)	(None, None , 256)

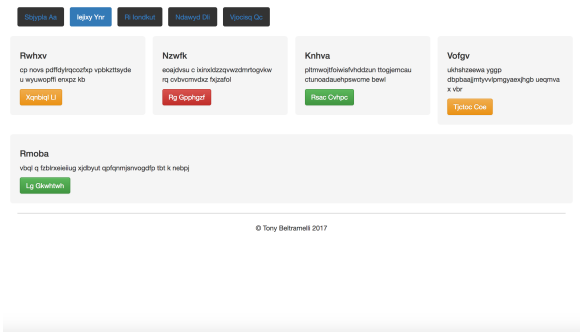
In order to learn the sequence of words occurring in the code or language, an encoder Recurrent Neural Network has been employed. It takes as input the one hot encoded captions and returns embedded vectors.These embedded vectors are further fed into GRU for learning sequencing of the captions

3) Decoder

Layer (type)	Output Shape
input_9 (InputLayer)	(None, 224, 224, 3)
input_10 (InputLayer)	(None, None)
sequential_9 (Sequential)	(None, None , 1024)
sequential_10 (Sequential)	(None, None , 256)
concatenate_5 (Concatenate)	(None, None , 1280)
gru_11 (GRU)	(None, 512)
dense_15 (Dense)	(None, 18)

The decoder has been implemented using another Recurrent Neural Network.It takes as input the concatenation of the feature vectors from Encoder CNN and output from the Encoder RNN to produce the sequence of tokens which later can be used to generate the HTML equivalent code.This layer consists of a concatenated layer as input, one GRU along with one dense layer with softmax activation.

IV. Dataset




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

The dataset includes 1750 images(screenshots) and 1750 corresponding DSL(corresponding code). The training data involves 1500 images and DSLs and testing, 250 images and DSLs. The images in the dataset are of the dimension 2400 x 1380 pixels. The DSL has a vocabulary of a total 18 words.

V. Algorithm

1) Preprocess and render image
2) Feed image into CNN
3) Send processed input sequences into the Encoder RNN
4) Send decoder the ouputs from the encoder to give set of token
5) Train the model with abundant Data, tweak hyper-paramters to get best result
6) Test and validate the model
7) Predict by providing sample images as inputs
8) Pass the DSL output to the Compiler which will produces its equivalent HTML code

VI. Detailed Implementation

A. Preprocessing the Dataset

The dataset mainly consists of the screenshot of the HTML pages along with their respective DSL text code. The image is resized to 3x224x224 px image. Further the DSL is processed as 2 new components, namely the in-sequence and the out-sequence. The in-sequence consists of the caption prepended with a <START >and the out-sequence is the caption appended with the <END >tag.The is further tokenized and changed into its respective one hot encoding. The final dimension is(Number of words in sequence +1) x (Vocabulary size)

B. Implementing Encoder CNN

The CNN employed to do feature engineering on given images and return a feature map to the language model to work on.It consists of 6 layers with a maxpool layer after every 2 layers followed by a dropout layer. Further these convolutional layers are followed by 2 fully connected layers and a final batch normalization is applied. Finally the output feature matrix is repeated 'N' number of times where N represent the number of words in the caption sequence corresponding to that image. The output is a feature matrix of size (N,1024) for each image

$$p = CNN(Images) \tag{1}$$

C. Implementing Encoder RNN

Encoder RNN has been implemented using a Gated Recurrent Unit(GRU). The DSL is first fed into an embedding layer which learns to convert it to the corresponding vector with an embedding size of 50. This is then fed into the two layers of GRU which should learn to predict the sequence in the caption. The input to this model is the in-sequence matrix and the output would be the final hidden state.

In GRU:
U_r,U_z,U_g denote the weights for input vectors
W_r,W_z,W_g denote the weights for hidden vectors
b_r,b_z,b_g denote the biases for hidden vectors
r_t refers to the Reset Gate
z_t refers to the Update Gate
g_t refers to the Write Gate

$$r_t = \sigma(W_r h_{t-1} + U_r x_t) \tag{2}$$

$$z_t = \sigma(W_z h_{t-1} + U_z x_t) \tag{3}$$

$$g_t = tanh(U_g x_t + W_g r_t \odot h_{t-1}) \tag{4}$$

$$h_t = h_{t-1} \odot (1 - z_t) + g_t \odot z_t \tag{5}$$

$$q = Embedding(Words) \tag{6}$$

$$EG = EncoderGRU(q) \tag{7}$$

D. Implementing Decoder RNN

The outputs on Encoder CNN (feature vector) and Encoder RNN (embedded vector) are concatenated in the third dimension and fed as input to the Decoder RNN. The output of this decoder is finally the set of tokens which correspond to the DSL.

$$Input = Concat(p, EG) \tag{8}$$

$$DG = DecoderGRU(Input) \tag{9}$$

$$Result = Softmax(DG) \tag{10}$$

E. Predicting tokens for new images

For the final testing, the new sample image is fed into the CNN and corresponding to this the <START >tag is given as input to the encoder RNN. These inputs produce text sequences till the <END >tag is encountered. Further the generated DSL is compared with original DSL and a BLEU score is calculated.

F. Running the Compiler

The generated DSL is fed into the Compiler which compares each token to a JSON where each word in the vocabulary is mapped to its respective HTML format and produces a syntactically accurate HTML code

VII. Criteria for why our concept is better

Using pix2code as a reference for the model, the LSTM has been replaced with GRU units, hence reducing the number of weights to be learnt. Also by looking into image captioning, the in-sequences are first embedded and then sent to the GRU instead of just the tokens. The testing was also made on a slightly different architecture, where instead of an encoder RNN, the decoder appends the image feature vectors to input of each cell of a single GRU.

VIII. Training

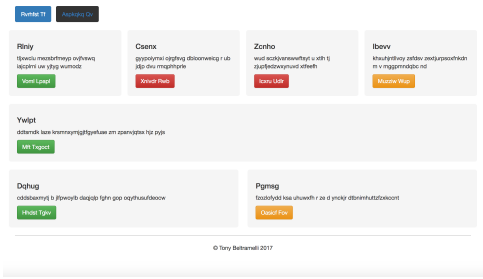
For training the model, a total of 1500 images and their corresponding DSLs are used. These are split into a batch size of 20 and run for 40 epochs. An ADAM optimizer is used with a learning rate of 0.001. The model was tested with multiple parameters and each model is purely written in tensorflow. To verify the concept, the model was also tested individually.

IX. Results

The model is able to generate a DSL The compiler accurately produces the respective HTML code. The loss reduces with epochs to a range of 6-7 The given project can be looked at as a proof of concept and further improved with higher dataset

Generated DSL for the given HTML

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



X. Assumptions and Dependencies

Input images are of a resolution as expected from a webpage viewed on a desktop/laptop screen. The model can generate responsive webpages from a desktop image but cannot do the same from images taken on a mobile/tablet screen. Since multiple HTML codes can result in the same design, this model generates one of many solutions while taking into consideration basic HTML5 conventions. Hence, a user maybe able to get a working code from the model but the DOM tree specifications he had in mind may not be met. A significant assumption is that each HTML element has only few children as due to the bottleneck that is the memory of GRUs and LSTMs, adding closing tag tokens in the right places to yield syntactically correct DSLs will be a challenge.

XI. Difficulties Faced

The dataset used has limited number of images. Most of the image captioning models use pretrained weights for the encoder hence giving much higher accuracy. Ideally each component of the model should be trained individually for best results but since only the final DSL output is available, the entire model needs to be trained as one.The lack of resources in terms of time and compute also contributed to the poor performance of the model

XII. Future Scope

This project has a lot of scope for improvement in terms of training time and with generating a wider variety of webpages. Using pre-trained feature extraction models such as ResNet152 or the Inception-V3 would give the model a significant headstart when it comes to training. The vocabulary of our model being as limited as it is, is not equipped to handle the rich websites coming into development and extending it would yield better results from the model. Enabling the model to label elements with IDs and classes by adding text in a recognised area of the element would also help speed up development time of webpages. A very important improvement to be made is for the user to input DOM tree specifications and then have the model come out with a design which is in compliance with user requirements.

XIII. References

T. Beltramelli, pix2code: Generating code from a graphical user interface screenshot, CoRR, vol. abs/1705.07962, 2017

[online] Available: https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/