Full Convolutional Front-End Code Generation Based on Web Page Screenshot

Zihao Chen[1,a] and Xiao Li[1,b]

[1]*Faculty of Information Technology, Beijing University of Technology, Beijing, China*
a. chenzihao000@qq.com, b. 382052461@qq.com

Keywords: Code generation, image caption, CNN, front-end development, DSL.

Abstract:Nowadays, Internet has gained its popularity, so more and more kinds of web applications need to be developed. For that, software company require a generator to output front-end code immediately based on web page screenshot to Improve development efficiency. Using full convolutional algorithm which contains CAE and CNN, we proposed a method performs better than pix2code. Our method follows the encoder-decoder framework and add hierarchical attention mechanism into it which learn from image caption methods. We use the PixCo dataset to complete the experiment.

1. Introduction

Nowadays, Internet has step into a whole new era, as Internet gaining its popularity, more and more people have become web application user, and it's an important goal to upgrade user experience on front-end. Regularly, a company or a software development team develop a front-end project by a series of step: first, product manager design prototype based on user demands; second, designer finish UI design based on prototype; last, front-end software engineer complete web application development. These steps usually come with long period and requirement of large team, but for those small company, they don't have such kind of resources, thus a font-end code generator is needed.

Beltramelli's pix2code[1] is the first research on generating HTML code based on web page screenshot. It proposed an encoder-decoder architecture, which using a CNN-based vision model and a LSTM-based language model as encoders, and a LSTM-based decoder. This method doesn't generate HTML code directly, they design a Domain Specific Language (DSL)[8] just for their task. DSL's is still a kind of programming language, so it's more accurate and concise than natural language. Lots of highly abstract development methods such as model-driven development and software factories are performed using DSL.

Pix2code still has some limit issues. First of all, its decoder is based on LSTM, which is a variant of RNN, that is to say, its training and inference must run by time step, so it would consume more time and computing power cannot be fully used. Secondly, RNN-based neural network is lack of ability to process structured information, but their DSL has a structured feature of hierarchical nesting.

To improve generating effectiveness and accurateness, we proposed a new method to generate front-end DSL with web UI design image. Our method uses full convolutional network on both encoder and decoder modules. We combine CNN and Auto Encoder as Convolutional Auto Encoder

(CAE) to complete feature extraction. And for language, we use one-hot encoding and word2vec to train word embedding. At last, we proposed a CNN-based decoder with hierarchical attention module.

2. Related Work

In recent years, code generation technology has gained widespread popularity, but automatically generating code based on images is still an unpopular subject. Software such as *OptimalJ* and *BorlandC#* Builder Enterprise, which appeared in the early 21st century, are relatively early model-based automatic code generation systems. In the current explosive development of the Internet, code generation has a large number of application scenarios on the front and back ends, for example, NHibernate is used to generate Object Relational Mapping (ORM) between Java language and database SQL on the server side. *Vue-cli* and *create-react-app-cli* are template-based command line scaffold. *Taro* and *mpvue* as model-based tools, can form an abstract syntax tree based on web-side code and generalize it into mini program, Android code or IOS code.

However, generating natural language based on images is a hot research area which is called image captioning. We can find common ground easily on these two kinds of tasks, that is, both of them describe objects and their relationship in images with one kind of languages, the only difference is our task use DSL and image captioning task uses human language. Image captioning is to understand the visual content presented by analyzing the image data, and to generate the corresponding reasonable language description. Image description belongs to the cross-technical field of computer vision and natural language processing, and is a task for performing multi-modal conversion from image to text. The above task can be given a definition: Given a two-tuple (I, C), where I represents the original image, C represents the text describing the original image, and the multimodal mapping $I \rightarrow C$ of image I and describing text C is the model we want.

Vinyals *et al*[2] propose NIC model in their paper, using encoder-decoder framework. They introduce CNN to encode images data, and LSTM to decode and generate sentences that describe images. Xu K *et al*[3] proposed a new model architecture, which introduce soft and hard attention to caption, and use visualization to understand the effect of attention mechanism. But they still use CNN+LSTM structure. In their RNN-based encode methods, they process a sentence from the beginning to the end, and input one word at a time step. The encoder calculates a hidden state every time step as input of next time step, so they can "remember" former words. But the truth is, we already knew every single word in the sentence during the training phase, a parallel computing method would be more efficiency. Aneja *et al*[7] proposed the first CNN+CNN structure, they use Gated Linear Unit (GLU)[9] as activation function instead of ReLU, in order to introduce gating mechanism in CNN.

In this paper, our proposed method is a front-end DSL generator with encoder-decoder framework. We won't use CNN+LSTM structure as was proposed in pix2code, instead, we use full convolutional structure for this task and introduce hierarchical attention mechanism among convolutional layers to have a better performance.

3. Proposed Method

3.1.Overview

Our method uses an encoder-decoder framework and is consist of three modules: vision encode module, language encode module, and decode module. The structure is shown in the Figure 1:

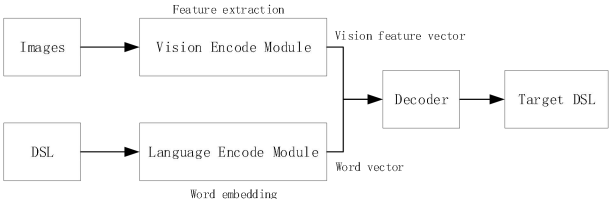

Figure 1: Architecture diagram.

3.2. Vision Encode Module

Auto Encoder [10] is a neural network designed to copy their input to the output. They work by compressing the input into a latent-space representation and then reconstructing the output of this representation. We use a convolutional layer as hidden layer of Auto Encoder and get Convolutional Auto Encoder. The principle is the same as that of an Auto Encoder. It down-samples the input symbol to provide a smaller potential representation, and forces the Auto Encoder to learn a compressed version of the symbol. Structure diagram shows as follow in Figure 2.

However, when the quantity of nodes in hidden layer is more than input and output dimensions, Auto Encoder cannot learn features we expect, instead, it will directly copy input to output. So, we introduce sparse control to avoid this kind of situation. Improved equation of cost function and Kullback–Leibler divergence as follow:

$$J_{sparse}(W,b) = J(W,b) + \beta \sum_{j=1}^{S_2} KL(\rho||\hat{\rho}_j) \tag{1}$$

$$KL(\rho||\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1+\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} \tag{2}$$

ρ represent a constant near 0 as sparse parameter. β represent penalty factor used to control activity of hidden neural unit, and it also near 0. $\hat{\rho}_i$ represent one time output hidden layer node j.

We also add a noise layer after input layer to improve encoder's ability of noise reduction. We resize all images to 256×256 as input, convolution kernel is 3×3 and stride is 1. We learn from VGG16's[4] structure, the first two convolutional layer's width is 64, and another two layers with width of 128, then comes three layers of 256 cells, and three layers of 512 cells, at last there is two layers with 512 cells. We use max-pooling at pooling layers, pooling kernel is 2×2 with stride 2. Both input layers and output layers' width is 256. To train CAE, we randomly choose 10000 tiles which size is 8×8 in all 1750 pictures of dataset as basic components, in this way, we can initiate convolutional kernel. After vision encoding, every image would transform into feature vector.

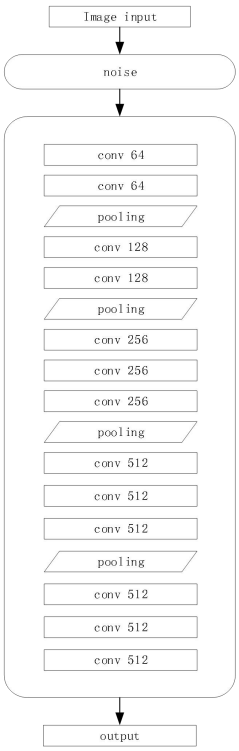

Figure 2: Convolutional Auto Encoder structure diagram.

3.3.Language Encode Module

We use the same DSL as was designed in pix2code, its grammar based on JSON which is frequently used by front-end developer, it shows a great advantage on presenting elements' hierarchical relationship. Braces present parent-child relationship and commas present sibling relationship.

To encode DSL we introduce one-hot encoding and word2vec method to process word embedding. The principle of one-hot encoding is to use the bit status register to encode each state, each state occupies a registered bit, only one of which is valid at any time. That means one-hot encoding maps discrete features to Euclidean space, making it possible to calculate similarity between original features. But it usually comes with dimension explode and it ignores the link between words.

Mikolov *et al* [5] proposed word2vec method which improve Neural Network Language Model (NNLM) [6] which came up with the concept word embedding. Word2vec is consist of two model: Continuous Bag-of-Words (CBoW) model and Continuous Skip-Gram model. The former learns the expression of the word vector by predicting the target word through the context, and the latter learns vectors of words nearby with the target word. CBoW can smooth the distribution of words, which is like regularization, and can provide very good performance when the input data is not too large. But the skip-gram model is more granular, so when we have a larger dataset, we can extract more information, so it is a more accurate word embedding method. Thus, we choose skip-gram to transform DSL to vector as language encoding.

Each word is specified as two d-dimensional vectors. Suppose a word is indexed in the vocabulary as i. When the word is used as the target word, the vector is expressed as $v_i \in \mathbb{R}^d$, and $u_i \in \mathbb{R}^d$ as the context word vector, ω_c with index c is the target word, w_o with index o is the context

word. The equation for calculating the conditional probability of a context word given a target word is as follows:

$$p(\omega_o|\omega_c) = \frac{\exp(u_o^T v_c)}{\sum_{i\in v}\exp(u_i^T v_c)} \tag{3}$$

We put special mark <START> at the beginning of every DSL files, and <END> at the ending, just to tell our program when to start and when to end as flags. After language encoding, every word would transform into word vector.

3.4.CNN-based Decode Module

Decoders based on convolutional neural networks use a feedforward method instead of a recurrent method using RNN. We use a simple feedforward deep neural network to predict the probability distribution. In order to be consistent with the description method based on the RNN decoder, although it is possible to operate in parallel in the training of CNN models with all known truth values, the "time step" is still used here to represent each word vector and graphic feature vector Decoding process. We divide this part of work into three modules: convolution module, attention module, and prediction module. Structure as shown in Figure 3.

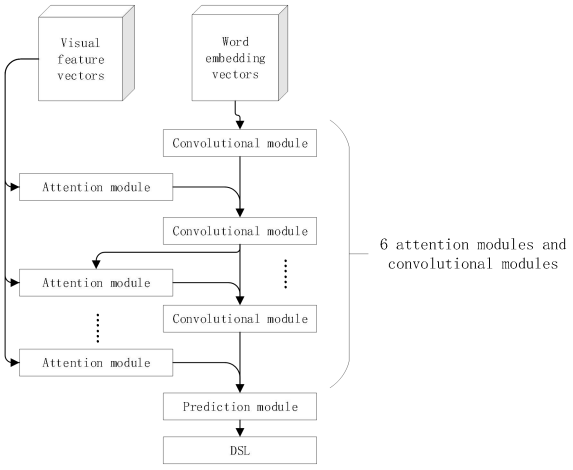

Figure 3: CNN-based decode module structure diagram.

3.4.1. Convolutional Module

In the RNN-based decoder, the prediction of each word will depend on all the words before the word entered at the current time step. Similarly, in the CNN-based decoder, we also need to prevent future word vectors from being used for convolution operations at the current time step, so we use masked convolution that only operates on past data.

Masked convolution is a convolution method proposed in pixelCNN [11]. Its role is to only retain pixels' data before the center point. $E = [e_1, e_2, ..., e_l]$ are the word embedding vectors output in the previous module, and its original content is a piece of DSL code containing l words. The activation function GLU is introduced in the mask convolution layer, and its equation is as follows:

$$h_l(X) = (X * W_l + b_l) \otimes \sigma(X * V + c_l) \tag{4}$$

W_l represents the weight matrix of the layer, b_l and c_l represent the offset of the layer, * represents the convolution operation, and ⊗ represents the element multiplication. GLU is different from ReLU which commonly used in convolutional neural networks. It is introduced after the gating mechanism of LSTM. Since convolutional networks do not suffer gradient disappearance which might occur in recurrent networks, there is no need to introduce forget gates.

The inference will start from the special mark <START> and sequentially perform each word of DSL code as input. The image feature vector is connected with the word embedding vector as the convolution input. After multi-layer masked convolution, the first time step of sampling is completed and inferenced the first word ele_1. Then ele_1 is input into the feedforward network to generate next word ele_2, and so on, until current word is the end mark <END>, or until the upper limit of the number of words we have reached. Since all truth values, that is, the words ele_1, ele_2, etc. are known, all time steps can be performed simultaneously.

3.4.2. Attention Module

Attention mechanisms are also introduced in this section. We connect the image feature vector with the word embedding vector of each layer. We calculate a separate attention parameter and a separate attention vector for each word. The calculation equation of the attention parameter and the image attention feature vector are as follow:

$$\omega_{ij} = \frac{\exp\left(e_j^T U v_i\right)}{\sum_i \exp\left(e_j^T U v_i\right)} \tag{5}$$

$$A_j = \sum_i \omega_{ij} v_i \tag{6}$$

Where e_j represents the j-th word embedding vector after convolution, v_i represents the feature vector at image i, U represents the linear layer applied to v_i, ω_{ij} represents the attention parameter, and A represents the image attention feature vector.

For different levels of DSL, the performance of convolution can actually obtain different attention information from image features, so we use the hierarchical attention mechanism. The hierarchical attention used in the RNN decoder is process one word vector followed by another word vector, which can be understood as horizontal attention. In this section, the attention module is added between every two convolutional layers, and there is also one before the final prediction module. This kind of layer-to-layer approach can be called vertical attention. This can effectively prevent lateral connections at the same layer, and can also train the entire DSL in parallel. Under the function of the GLU, the overall calculation equations are as follow:

$$A_j^{l-1} = \sum_{i=1}^{N} \omega_{i,j}^{l-1} v_i \tag{7}$$

$$h_a^l = W_a^l * h^{l-1} + W_a^{att} * \left[a_j^{l-1}\right]_{j=1}^{L} + b_a^l \tag{8}$$

$$h_b^l = W_b^l * h^{l-1} + W_b^{att} * \left[a_j^{l-1}\right]_{j=1}^{L} + b_b^l \tag{9}$$

$$h^l = h_a^l \otimes h_b^l \tag{10}$$

ω can be calculated from (4), l represents the number of layers and W represents the weight matrix.

3.4.3. Predict Module

This module consists of a neural network with a hidden layer. It takes the attention feature vector output from the previous step and the convolutional content vector as inputs. Equations as follow:

$$h_j^p = f(W_a^p A_j + W_c^p C_j + b^p) \tag{11}$$

$$P_{j+1} = softmax(U^p h_j^p) \tag{12}$$

The formula represents the operation of processing the word vector at position j in the sentence, $I_j^{attention}$ and C represent the attention feature vector and the convolution vector, W represents the weight matrix, and the function of the f function is $f(x)$=max(x, 0.1x). This module uses a 1×1 convolution kernel for the final prediction operation.

3.5.Training

We use a cross-entropy loss function to calculate the probabilities of CNN modules. Each pair of data is composed of a PNG format image and a DSL code file. We use conv5_3 on attention module and size of prediction module's hidden layer is set to 1024 which λ is 1e-5. All images are resized to 224×224 before training or inference in our program. We introduce *RMSProp* to optimize and the learning rate is start with 1e-5 and after every 15 epochs it will decay 10%.

4. Experiments

We chose PixCo from pix2code to be our dataset. It contains 1750 pairs of web page screenshot and corresponding DSL file. We separate it to 1500 pairs for training and 250 for testing. We operate 4 experiments in this section and 3 of them as baseline of the method we proposed. CNN+LSTM is the original pix2code experiment. CAE+LSTM is an upgrade version of pix2code which change former vision module into Convolutional Auto Encoder. CAE+CNN is our method without attention modules, and the last experiment is out method. All of these experiments are operated in a cloud server with Nvidia GTX1080 GPU and Intel Core i7 CPU and Ubuntu 16.04 LTS operating system.

```
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}}
```

Figure 4: Image and DSL pair of PixCo dataset.

Table 1: Accuracy and error result.

Method	Accuracy							Error (%)
	BLEU1	*BLEU2*	*BLEU3*	*BLEU4*	*METEOR*	*ROUGE*	*CIDEr*	
CNN+LSTM	0.8149	0.7581	0.6643	0.5821	0.4678	0.7254	3.0455	12.57
CAE+LSTM	0.8226	0.7675	0.6720	0.5874	0.4697	0.7074	3.1504	12.76
CAE+CNN	0.8395	0.7824	0.6896	0.6014	0.4475	0.7105	2.8745	12.23
CAE+CNN+Attention	0.8742	0.8251	0.7163	0.6285	0.4895	0.7547	3.3569	10.63

We compared CNN+LSTM method and CAE+LSTM method. Based on experimental result, we found that after we add Auto Encoder, BLEU1 improved by 0.9%, BLEU2 improved by 1.2%, BLEU3 improved by 1.1%, BLEU4 improved by 0.9%. The results show that CAE improves the quality of the visual features to some extent, thereby improving the recognition accuracy, but the effect is not obvious. We compared CAE+LSTM method and CAE+CNN method, it shows that after CNN decoder is introduced, BLEU1 improved by 2%, BLEU2 improved by 3.2%, BLEU3 improved by 2.6%, BLEU4 improved by 2.3%. The results show that the CNN decoder improves the accuracy slightly, but it takes 0.35 seconds per batch for the one with CNN decoder and 1.04 seconds per batch for the one with LSTM decoder (batch size is 32), which means CNN-based method is indeed trained faster than RNN-based method. We compared our methods with or without attention mechanism, it turns out that the full version of our algorithm has obviously higher accuracy and lower error rate than each of other versions.

5. Conclusions

In this paper, we proposed a CAE+CNN framework with vertical attention mechanism to deal with the problem of generate DSL code based on web page screenshot, and it proved to be a more efficient and accurate method.

References

[1] Beltramelli, Tony. "pix2code: Generating Code from a Graphical User Interface Screenshot.".

[2] Vinyals, Oriol, et al. "Show and tell: A neural image caption generator." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[3] Xu, Kelvin, et al. "Show, attend and tell: Neural image caption generation with visual attention." International conference on machine learning. 2015.

[4] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[5] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

[6] Bengio, Yoshua, et al. "A neural probabilistic language model." Journal of machine learning research 3.Feb (2003): 1137-1155.

[7] Aneja, Jyoti, Deshpande, Aditya, and Schwing, Alexander. "Convolutional Image Captioning.".

[8] Van Deursen, Arie, Paul Klint, and Joost Visser. "Domain-specific languages: An annotated bibliography." ACM Sigplan Notices 35.6 (2000): 26-36.

[9] Dauphin, Yann N., et al. "Language modeling with gated convolutional networks." Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017.

[10] Ng, Andrew. "Sparse autoencoder." CS294A Lecture notes 72.2011 (2011): 1-19.

[11] Van den Oord, Aaron, et al. "Conditional image generation with pixelcnn decoders." Advances in neural information processing systems. 2016.