Generating Code from CSS Animation

Huifeng Chen
NYU Courant
hc2489@nyu.edu

Nicholas Wen
NYU Courant
nw683@nyu.edu.org

Abstract

The goal of this study is to investigate the ability for a Convolutional Network (CNN) and Long Short Term Memory network (LSTM) architecture to take a video clip of an animation and generate the corresponding CSS code. The challenge is a two-fold problem of animation recognition and code generation. In our investigation, we experimented on ResNet architecture with 3D convolution on image frames and optical flow for action recognition and seq2seq framework for code generation. On our custom dataset, we achieved over 98% accuracy for animation classification and reasonable results for the code generation.

1. Introduction

Program synthesis is a field of interest in the Artificial Intelligence community that focuses on creating algorithms that understand complex procedural behaviors and produce code. In a standard development process, a developer is given a specification which includes: inputs, expected outputs, performance requirements, security protocols, etc, which they would use to implement a program. The development process will often include multiple iterations of refactoring, debugging, and feedback in order to achieve a satisfactory product. The aspiration for program synthesis is to substitute the need for a programmer to do the development process and automatically produce working code that meets the specified requirements.

Implementing CSS code based on a visual example is one of the tasks that would benefit from program synthesis. As a domain specific language, CSS is composed of rules and parameters that describe appearances and animation effects of HTML elements on a webpage. Given screenshots and specs generated by animation softwares, a web developer may be able to eyeball and estimate the animation type and parameters, but it is not trivial to correctly recover complicated animations involving multiple rules. Program synthesis is useful in instances where the source code is not easily available for viewing or the desired effect was created for demoing purposes by something other than CSS, such as Flash or video editing software.

In the field of computer vision, video captioning attempts to classify actions in videos and produce a human readable sentence that describes the video. Common approaches involve using a CNN to perform object and action classification, then passing the feature maps extracted into a LSTM to generate natural language. We noticed the similarities between the task of video captioning and our goal of generating code from video media. We pursued using a deep learning architecture often used for video captioning and extrapolating it to perform program synthesis for CSS code.

For this task, we have produced a dataset composed of a variety of HTML elements (text, images, and shapes) being animated by CSS rules. We limited the problem space to cover limited set of animations applied to a single HTML element. We hope this work can be the first steps towards exploring this approach and the results learned from it can be applied to more complicated animation videos. The implementation is here[1] for reproducing.

2. Related Works

There is no previous work done on synthesized animation recognition. There are three parts related to this work, video action recognition, video captioning and neural code generation.

Karpathy et al. [5] trains 3D CNNs with Sports-1M dataset and experimentally found that 3D convolution kernel is the most effective. Carreira et al. [2] shows that CNNs with spatio-temporal 3D convolutional kernels are more effective than 2D CNNs for the action recognition task. Simonyan et al. [7] combines dense optical flow and RGB images to provide better features, and achieve higher accuracy on action recognition.

For image captioning, Kulkarni et al, [6] generates descriptions by filling sentence template slots with words(attributes, preposition) selected from a conditional random field that predicts the most likely image labeling. Vinyals et al.[10] proposes a encoder-decoder framework

[1] https://github.com/josherich/Animation2Code

that extracts CNN features and feeds them into a RNN decoder, in order to generate captions on the output feature maps.
For image to code generation, Ellis et al.[3] shows a variety of techniques used to constraint solution space and guide code generation. There are also efforts to generate code from static webpage screenshot using a image captioning framework. Beltramelli et al.[1] uses a LSTM to encode HTML code and with the output of a CNN, feed to another LSTM encoder to generate domain specific language. The compressed domain specific language (DSL) is manually defined to ease LSTM encoding.

3. Approach and models

We approach this problem in two steps, first to train a neural network on animation classification task using a custom dataset, and extract temporal spatial features from last linear layers of the network. The second is to train a seq2seq[9] model with: features maps from the CNN and code embeddings as input. The entropy loss between RNN output and gold code text is computed to back propagate, the training directly maximize the probability of correct code text given the video feature vector:

$$\theta^{\star} = \arg\max_{\theta} \sum_{A,C} logp(C|A,\theta)$$

Training: In classification task, the hidden layers in the CNN learn representation of the videos in fixed-length vectors. In the second task, there are two differences with classical image captioning, first, one-hot-encoding vector is used to encode code text instead of pretrained word embedding. Second, video embedder instead of single image embedder is the input of RNN encoder. We believe these differences do not significantly impact the method's compatibility with our problem.

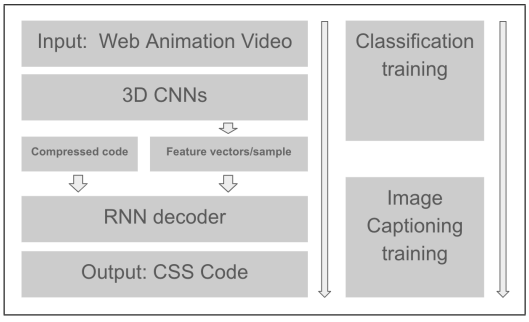

Figure 1. Model architecture in training time

Inference: At **test time**, CNN receives new animations frames and outputs 512 dimension features vectors. The trained RNN decoder receives representation of video features, and outputs prediction code using beam search.

4. Dataset

There is no previous work on this specific problem, we created our own dataset of 1.2k videos of various types of web animations, including the classification labels and according code text for each clips.

4.1. Classification setup

For the ease of covering most types of web animation, we use a popular CSS library animate.css[2] as the starting point of defining animations classes. The library include 77 types of animation, some examples are: boundIn, bounceOut, fadeIn, heartbeat, etc... We build our dataset by performing the animations at various speeds and on different objects. We iterate 4 types of speed: slower, slow, fast, faster, which correspond to animation duration of 3s, 2s, 800ms, 500ms. Each animation is performed on four kinds of objects: line, square, image, and text. The combinations of animations, speed, and object results in 16 different clips for each of 77 animation classes. This data synthesis setup produces 1232 video clips in total.

4.2. Generating video clips

We first generate HTML files for each training sample, using HTML templates. To efficiently generate the video clips as it is in browsers, we use a display emulator library Xvfb[3] integrated with ffmpeg[4]. The display simulator runs an instance of the Chrome browser, and ffmpeg screencasts the output of display for 5 seconds, in a framerate of 24/s. A video with 120 frames, of size 256x256 is generated for each combination of animation type, speed, and object.

4.3. Code compression

For efficient training and to optimize learning with seq2seq model, we compress CSS code text manually by replacing the tokens with unique abbreviation. The max length of original code in training set is 301, mean length 118.7. After compression, the max length is 162, mean length 69.3. The vocabulary size across the dataset is 184, including keys and values in declarative statements.

4.4. Dataset partition

For the dataset split, we randomly pick 60 out of 12k videos as validation set. For the testing set, 60 out-of-domain CSS animations are manually crafted, by changing the parameters in animation function of samples in training set. The manual test samples keep the appearance of the animations.

[2]https://daneden.github.io/animate.css/
[3]https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml
[4]https://www.ffmpeg.org/

4.5. Dataset format

We follow the data format and annotations of UCF101 [8], for the convenience of using existing training codebase. The preprocessing steps follow [4]'s setup: one image is generated for each frame. For each sample video, 120 images are used as the input for CNN.

5. Experiments

We experiment the model architecture in two parts: the 3D CNN used to classify the animation type and the LSTM that generates the code. For the task of classification, we tested ResNet 10, 18, and 34 (the results are shown in Table 1). The three models achieved very similar results with more than 98% accuracy on the training set and 87% accuracy on the validation set. However, the difference between the validation and training accuracy suggests over-fitting on the training set. This is expected since the visual variance of web animation frames in our dataset is much smaller than real world images and videos, and our dataset size is one order of magnitude smaller than UCF101[8] who has 13k clips.

5.1. Two Stream

In order to address the issues with left right orientation, we evaluated our data using a two-stream framework for video recognition [11]. Using the dense_flow project[5], we extracted the optical flow from our dataset and converted the 5 second videos to 119 frames representing the x and y flow. Then using a Pytorch implementation of a two stream framework designed for UCF101 as a base[6], we trained ResNet18 using the optical flow.

When evaluating the two stream framework, we tested two different video lengths: 10 and 70 consecutive frames. We trained the ResNet18 model using 10 consecutive flow frames per video, which resulted in an accuracy of 68% on the training set. The significantly lower accuracy rate is expected when considering that the first 10 frames represent 0.4 seconds of a video and some animations taking up to 3 seconds to completely execute.

Therefore, we increased the number of consecutive frames to 70 consecutive flow frames (representing 2.8 seconds of the video) and achieved up to 98% training accuracy and 80% validation accuracy, which is very similar to the results from the 3D CNN.

Our validation set at this point is 20 examples, which covers 20 of the 77 classes. So we decided to increase the validation data by randomly sampling 2 examples for each class from the training set. Based on the new data split of 1078 training examples and 154 validation examples, we achieve 98.2% training accuracy and 90.9% validation accuracy on ResNet18.

[5]https://github.com/yjxiong/dense_flow/

[6]https://github.com/bryanyzhu/two-stream-pytorch

Method	Training	Validation
ResNet10	99.01	89.67
ResNet18	97.61	87.50
ResNet34	99.83	89.17

Table 1. CNN accuracy training on Animation2Code dataset

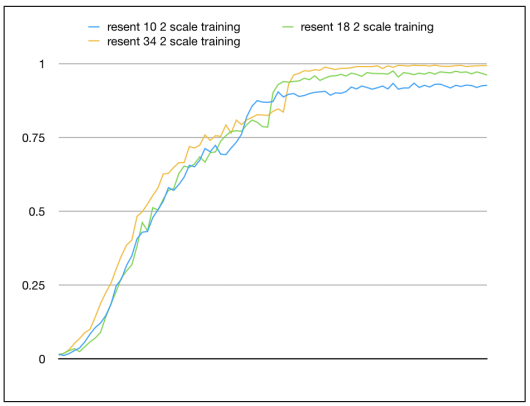

Figure 2. training accuracy of ResNet with 10,18,34 layers

Figure 3. validation accuracy of ResNet with 10,18,34 layers

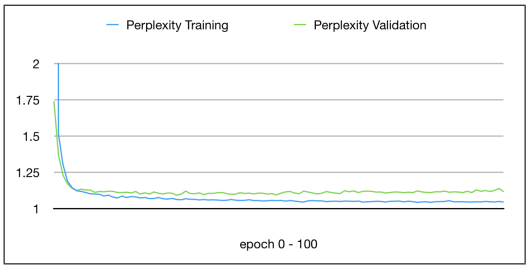

Figure 4. Perplexity of LSTM training epoch 0 to 100

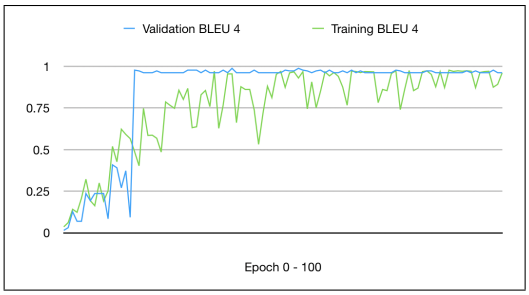

Figure 5. Bleu 4 score of LSTM training epoch 0 to 100

5.2. Seq2seq training and overfitting discussion

Figure 4, 5 show perplexity and BLEU 4-gram score progress during the training of 100 epoch. For seq2seq training, the total number of parameters is 786k with an input vector of size 256 for video feature vector and 256 for word embedding, and output vector of size 256. Since the total number of tokens and feature vectors in training set is of size 614k, it's possible for the network to overfit on the training dataset. In the future, we would create dataset with one order of magnitude bigger size and out-of-domain test set to validate how much seq2seq training can generalize on such domain specific language generation.

6. Error Analysis

On ResNet10, for the best validation epoch, we had a 76 mislabels (spanning 25 classes) out of 1212 total training examples. We noticed that 58 error cases of the 76 error cases were related to mistaking the directional orientation of the animation (ex. zoomInLeft being mistaken for zoomInRight). The animation type (ex. zoom, bounce, fadeIn, fadeOut) were correct for all 58 cases, but specifically the left right classification is wrong - there was 1 up down error. For the 17 other errors, most errors were 1 off mistakes belonging to different classes. However, we noticed 3 counts of "heartbeat" being mistaken for "tada" and 3 counts of "fadeIn" being mistaken for "lightspeedIn".

6.1. Seq2Seq

For seq2seq training, we experiment with 1, 2, and more layers, we get the best result from 1 layer and similar performance with 2 or more layers. We manually check code execution in the browser, 18 out of 20 in validation set successfully generate an animation effect on an HTML element. 4 out of these 16 predict the correct speed, and 5/16 predict the correct object animated, 12/16 recover the basic appearance of the animation. Below is one prediction result(compressed):

> **Prediction:** @keyframes { from { op : 0 ; tf : t3 (0 , 2000px , 0) ; } to { op : 1 ; tf : t3 (0 , 0 , 0) ; } } sel { ad : fast ; pat : image ; }

> **Goal:** @keyframes { from , 60% , 75% , 90% , to { atf : cb (0.215 , 0.61 , 0.355 , 1) ; } from { op : 0 ; tf : t3 (0 , 3000px , 0) ; } 60% { op : 1 ; tf : t3 (0 , -20px , 0) ; } 75% { tf : t3 (0 , 10px , 0) ; } 90% { tf : t3 (0 , -5px , 0) ; } to { tf : t3 (0 , 0 , 0) ; } } sel { ad : slow ; pat : square ; }

The type of this animation is "bounce in up" on square with slow speed, the prediction recovers the change of opacity and "translate3d", but it fails to recognize the more complex animation keyframe specs. The model has a low accuracy predicting numerical values since it's treated as text in our training.

7. Future works

Without an out-of-domain test set, we are not able to prove this end-to-end training generalize and generate code directly on video clips of web animation. In the next steps, we would want to increase our dataset from 1232 examples and increase the ratio of examples validation/test sets and the training set. By introducing more types of animations and varying parameters in animation specs, we would be able to increase the dataset size by one order of magnitude.

For RNN decoder, the code generation suffers from the problem that high accuracy does not ensure the code executes. The size of the code corpus in the training dataset is insufficient for the RNN decoder to learn a stable language model on CSS DSL. An approach we would like to try is: to use a web scraper to acquire a large dataset CSS code from Github and pretrain a language model on it. The numerical value in the animation specification is not properly handled in the current implementation, it requires special neural components to inference numerical parameters for functions.

8. Conclusion

We proposed using a 3D CNN and RNN decoder to convert a video into CSS code. We achieve classification training accuracy above 98% for ResNet10, ResNet18, Resnet34, and ResNet18 with the two-stream framework. Even though our results seem promising, we noticed an accuracy gap of around 10% between the training and validation sets for all versions of ResNet, which suggests that our model is overfitting on the training set. For seq2seq training, our model has success producing general animations, but has low accuracy predicting numerical values, such as opacity and rotation specs. In general, we need to expand the size and diversity of the dataset, as well as test on a more extensive out-of-domain test set to better prove the effectiveness of network architecture producing code from synthesized animations. We hope this work can be a first step in exploring the use of CNNs and LSTMs to perform code generation for animations.

References

[1] T. Beltramelli. pix2code: Generating code from a graphical user interface screenshot. *arXiv preprint arXiv:1705.07962*, 2017. 2

[2] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. 1

[3] K. Ellis, D. Ritchie, A. Solar-Lezama, and J. B. Tenenbaum. Learning to infer graphics programs from hand-drawn images, 2017. 2

[4] K. Hara, H. Kataoka, and Y. Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6546–6555, 2018. 3

[5] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, June 2014. 1

[6] G. Kulkarni, V. Premraj, S. Dhar, S. Li, Y. Choi, A. C. Berg, and T. L. Berg. Baby talk: Understanding and generating simple image descriptions. In *CVPR 2011*, pages 1601–1608, June 2011. 1

[7] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos, 2014. 1

[8] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild, 2012. 3

[9] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks, 2014. 2

[10] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015. 1

[11] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao. Towards good practices for very deep two-stream convnets. *CoRR*, abs/1507.02159, 2015. 3