Institute of Systems Science, National University of Singapore, Singapore 119615

GRADUATE CERTIFICATE PATTERN RECOGNITION SYSTEMS PRACTICE MODULE REPORT

DrawFUN - Sketch Drawings Optimization System

TEAM MEMBERS

LIN YINGLIN

SONG BINGHENG

XIONG HUI

ZUO ZONGYUAN

MASTER OF TECHNOLOGY

CONTENTS

1. EXECUTIVE SUMMARY

Since childhood, human beings have developed the ability to use stroke-based sketch drawings to reconstruct the real world they see from their eyes and to express their nonrepresentational inner feelings. Nowadays, with the development of AI technology, the machine could think and act in daily routines like humans to some extent, as well as in sketch drawings.

Our team is comprised of 4 members who enjoy drawing in our daily lives. However, it is embarrassing that we often try to draw a stick figure, such as for a cat, and it turns out to be like a dog absurdly. Due to the fact that without systematic learning, not everyone has the talent to draw like professional painters. So, we hope to build a sketch drawings optimization system to help ordinary people to improve their drawings.

DrawFUN is such a fun, artistic and playful system. It can help users to continue their incomplete drawings as well as inspiring their artistic imagination. According to the content drawn by the user, the machine will perform artistic creation in the background, so as to randomly, continuously and creatively generate sketches of the

same category for the user's preference. In other words, by using the DrawFUN system to draw, users do not have to start from an extremely blank canvas with their own imagination. Instead, the machine is enabled to provide certain creative samples to the users according to what they are drawing for.

Inspired by the image processing techniques imparted to us in lectures, we build the system based on sketch classification and sketch generation. The programming languages we used are JavaScript and Python. Firstly, we develop a webpage to obtain users' drawing data from an easy-to-use UI. Then, the RNN (LSTM) model is utilized to identify and classify users' drawing content into more than three hundred categories, and finally the SketchRNN model is used to automatically generate sketches corresponding to the previous classification results.

We appreciate this experience of online collaboration, which requires more leadership, self-discipline and time management compared with usual offline group work. We acknowledge that this project is still immature and there are a number of aspects to be improved, but all of us spare no effort to solve every problem we encountered in the process of learning, and contributed to the final complete of the project.

2. PROBLEM DESCRIPTION

It seems that human lose imagination gradually while they are growing up. Because as people get older, they become more aware of the rule of the real world, accept more pressure and responsibilities in daily lives, and have less time to explore the space in which we live. For those who have not received systematic, continuous and authoritative artistic guidance, they may be too afraid to begin their first stroke in drawing board although they have certain interests in painting. Moreover, not only usual people encounter this situation in great possibility, even professional painters may encounter similar painful situations where they may lose inspiration.

2.1. PROJECT OBJECTIVE

DrawFUN is designed and developed to help people to complete their sketch drawings and expand their artistic imagination. It tries to not only guess what user draws, but also offers a set of randomly generated sketch drawings of the same object. These previously unseen creative samples could help people to continue their drawings, expand their imagination and break the cognitive limits of what has been known. For those who are good at painting, they will be considerably inspired by the DrawFUN system.

Hopefully we can offer happiness to users with unexpected creation.

3. TOOLS AND TECHNIQUES

3.1 QuickDraw dataset

We used QuickDraw dataset for both classifier and generator parts. It includes vector drawings obtained from Quick, Draw!, an online game where the players are given a particular object category randomly and then required to draw corresponding objects in a limited short time. It consists of more than three hundred categories of common objects, and each category is considered as a dataset of 70,000 training samples, in addition to 2500 validation and 2500 test samples.

3.2 Sketch Classifier

Sketch Classifier is an RNN-based recognizer which can distinguish the object category that the user tried to draw. The model will use a combination of several convolutional layers, bidirectional LSTM layers, and a Softmax output layer to classify the drawings; We also add some batch normalization layers and dropout layers to improve accuracy. The model structure is shown in Figure1.

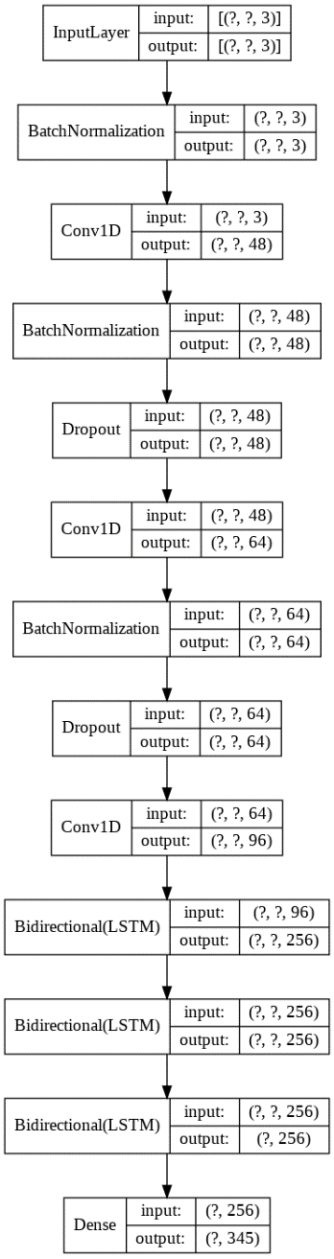

Figure 1: the classifier model

```
Model: "Classifier"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, None, 3)]         0
_________________________________________________________________
batch_normalization_18 (Batc (None, None, 3)           12
_________________________________________________________________
conv1d_18 (Conv1D)           (None, None, 48)          768
_________________________________________________________________
batch_normalization_19 (Batc (None, None, 48)          192
_________________________________________________________________
dropout_12 (Dropout)         (None, None, 48)          0
_________________________________________________________________
conv1d_19 (Conv1D)           (None, None, 64)          15424
_________________________________________________________________
batch_normalization_20 (Batc (None, None, 64)          256
_________________________________________________________________
dropout_13 (Dropout)         (None, None, 64)          0
_________________________________________________________________
conv1d_20 (Conv1D)           (None, None, 96)          18528
_________________________________________________________________
bidirectional_18 (Bidirectio (None, None, 256)         230400
_________________________________________________________________
bidirectional_19 (Bidirectio (None, None, 256)         394240
_________________________________________________________________
bidirectional_20 (Bidirectio (None, 256)               394240
_________________________________________________________________
dense_6 (Dense)              (None, 345)               88665
=================================================================
Total params: 1,142,725
Trainable params: 1,142,495
Non-trainable params: 230
```

Figure 2: the classifier parameters

The input is a drawing which is encoded as a sequence of strokes of points in x, y, and flag, where flag indicates whether the point is the last point in a new stroke. Then, a series of 1-dimensional convolutions is involved. Next, LSTM layers are applied and the sum of the outputs of all LSTM steps is fed into a Softmax layer to make a classification decision among the classes of drawings that we know.

3.3 Sketch RNN

SketchRNN is based on seq2seq automatic coding framework, uses hypernetworks as a repeatable neural network unit. The purpose of using a seq2seq encoder is to train the neural network to encode the input statement into a floating-point vector, that is called hidden vector, and then use this hidden vector and decoder to generate the output sequence, to reconstruct the input sequence as close as possible. The model structure is shown in Figure 2, and our load model structure is shown in Figure 3.

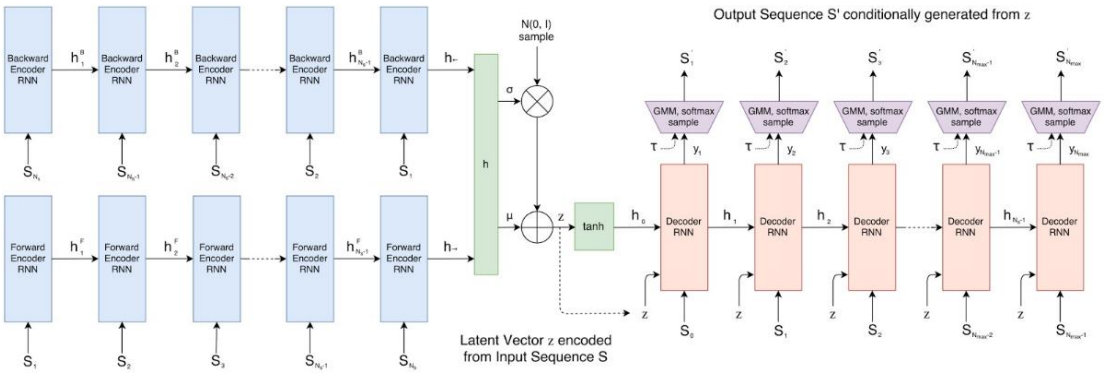

Figure 3: the SketchRNN model

Model: "sketchrnn"

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	[(None, 131, 5)]	0	
encoder (Functional)	[(None, 128), (None,	667904	encoder_input[0][0]
decoder_input (InputLayer)	[(None, None, 5)]	0	
initial_state (Functional)	[(None, 512), (None,	132096	encoder[0][0]
decoder (Functional)	[(None, None, 123),	1386107	decoder_input[0][0] encoder[0][0] initial_state[0][0] initial_state[0][1]

Total params: 2,186,107
Trainable params: 2,186,107
Non-trainable params: 0

Figure 4: our load SketchRNN model

In addition, when generate new sketch drawings, the randomness level can be controlled by using the temperature parameter to scale the softmax parameter of the classified distribution and the parameter of the binary normal distribution. Usually, the temperature would be set between 0 and 1. The closer the temperature gets to zero, our model would become more certain, which means the generated sample will consist of the most likely points.

3.4 TensorFlow and TensorFlow.js

TensorFlow is an end-to-end open-source machine learning platform. It provides a simple and flexible framework that allows users to quickly turn new ideas from concepts into code, then create advanced models and eventually release them.

TensorFlow.js is an open-source JavaScript library for training and deploying machine learning models in the web browser.

3.5 ML5.js

ML5.js is an open-source JavaScript library which include TensorFlow.js and some additional components to help machine learning for the web in your web browser easily, accessible and approachable. The ml5.js library provides access to a variety of machine learning algorithms/models in the browser, building on top of TensorFlow.js which uses the browser's GPU to run all the calculations, with no other external dependencies.

Generally, data collection is tough, and fancy method usually is slow, so solutions that use off-the-shelf models instead of going for fancy method would probably bring more efficiency to our work. There are various corporations train models that ml5.js provides, like PoseNet, YOLO, SketchRNN (which we used in our project), and many other classical machine learning models.

4. SYSTEM DESIGN AND MODELS

Our system could be basically decomposed into three main modules. The first module is transformation between drawing strokes and the list of points. The second module is a classifier to distinguish what kind of object user draw. The third module is a generator to generate sketch drawings automatically. The detailed system design is presented using an annotated inference structure diagram as shown in Figure 4.

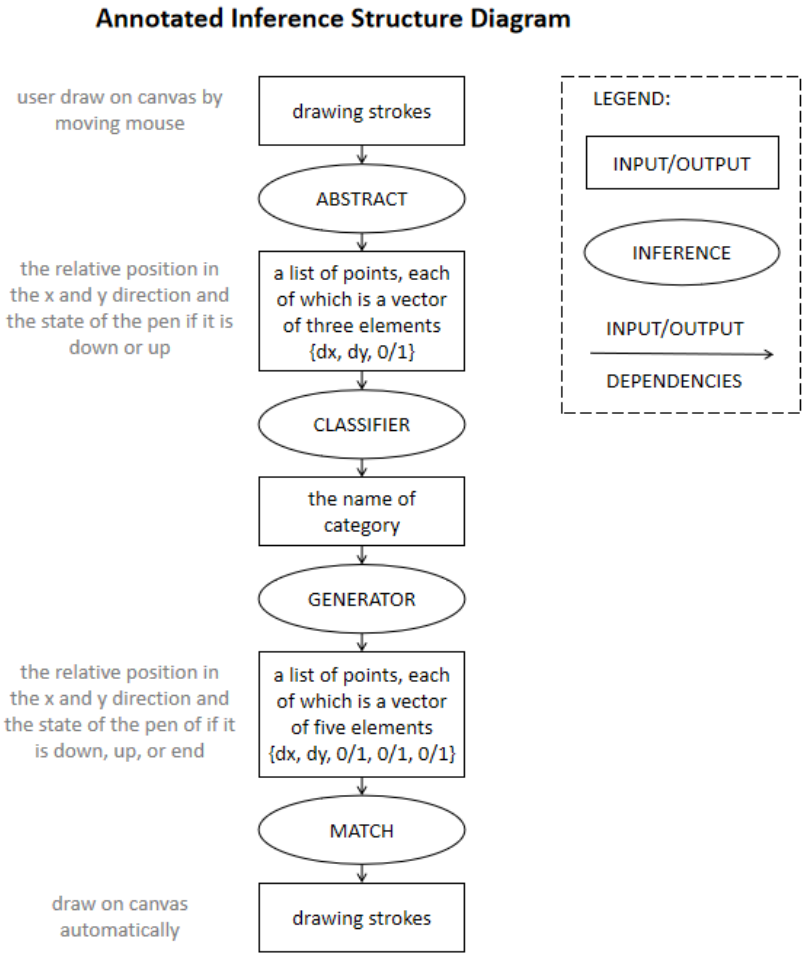

Figure 5: annotated inference structure diagram for the DrawFUN system

According to the system design, the appropriate model structure layout is defined, as shown in Figure 5. It specifies how the classifier part and the generator modules work in the whole system.

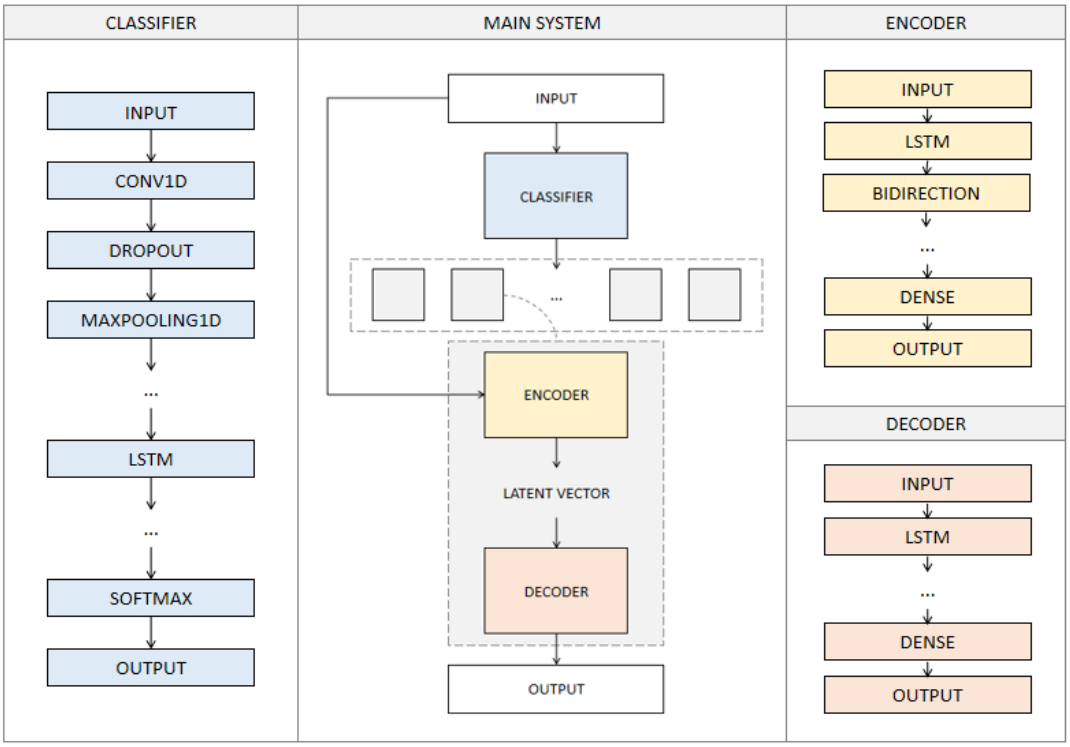

Figure 6: main model structure layout for the DrawFUN system

4.1. SYSTEM ARCHITECTURE

Although the above models are trained and tested using Python programming language, it is brought to life using JavaScript programming language in the form of a web-based graphical user interface which users can easily interact with. Figure 6, the system architecture diagram, illustrates how the application in the front-end has been interfaced with the back-end system.

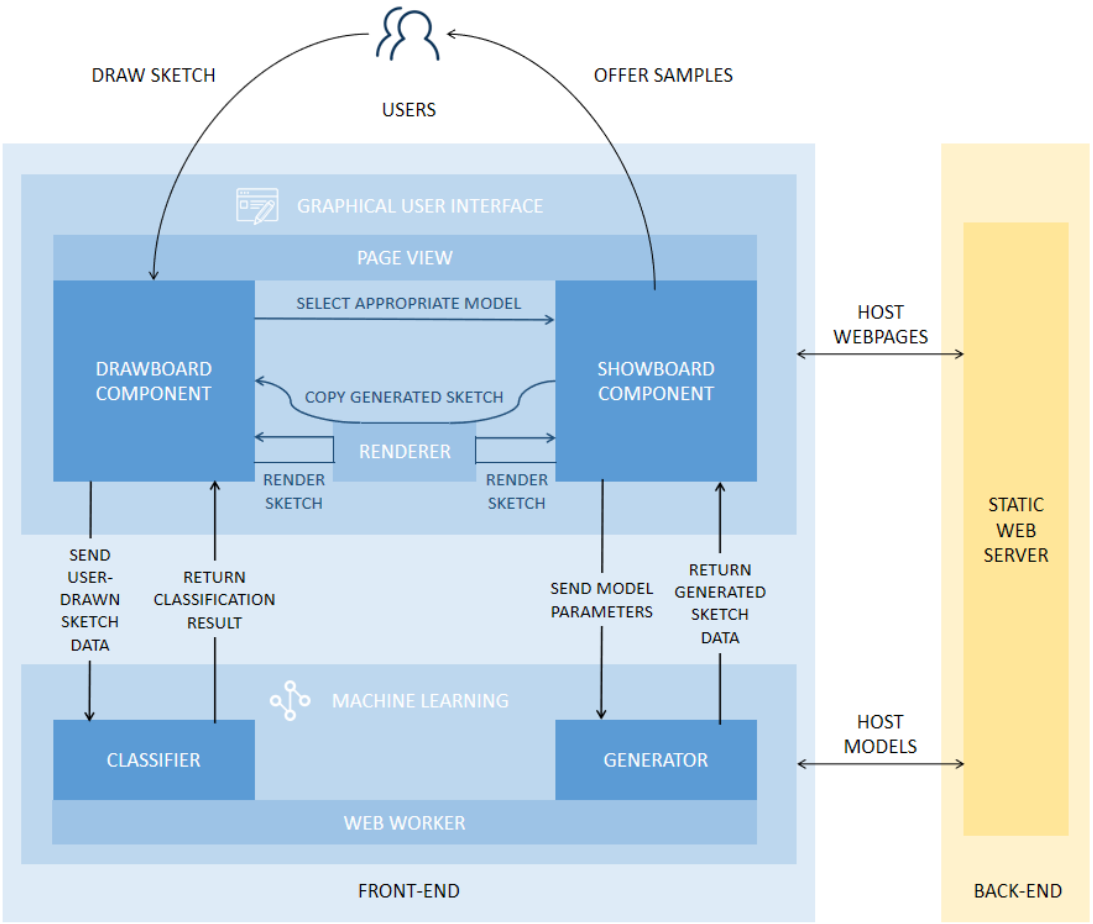

Figure 7: the DrawFUN system architecture diagram

The front-end of our system consists of two parts. The first part is the User Interface. Since the system is a single page web app, there's only one view in the UI part, which is composed by two components. The DrawBoard component is the place where user can draw sketch and the ShowBoard component will inspire user with samples generated based on what user drew.

The second part of the font-end is the Machine Learning part. The ML part runs on two background threads, aka Web Workers. Each worker contains a NN model. The Classifier will take the sketch data from DrawBoard component and determine what user drawn with a stateful RNN model. The Generator will then generate sample sketches based on the classification result.

The back-end of our system is a static web server, hosting the pages, scripts and models requested by the front-end.

4.2. SYSTEM IMPLEMENTATION

The front-end of the system is mainly developed under Node.js environment, utilizing

Vue.js and Bootstrap. The two Vue components, DrawBoard and ShowBoard, are all based on the HTML canvas element. DrawBoard component listens to user's mouse event, record the mouse position and mouse button state in form of a tuple (dx, dy, p), where dx and dy indicates the change between current position and previous position, and p is a binary value indicates whether the pen is down (0) or up (1). Each tuple, along with the coordinate of the starting point, will be passed in sequence to the renderer to do the final rendering job on the canvas element. Meanwhile, whenever the user finished one stroke (pen state changed from 0 to 1), the stroke data (an array of the above tuple) will be serialized and send to a Web Worker running on another thread, where the data will be deserialized and passed to the Classifier.

The Classifier uses TensorFlow.js to load the classifier model we built and trained previously in python, and utilizes the client-side CPU power (or GPU power, if the client supports HTML Offline Canvas API and runs on a GPU) to execute the model, which is so-called "offline prediction". The result of the Classifier will be again serialized and sent back to the DrawBoard component. After receiving the classification result, DrawBoard will emit an event carrying the result, which will be proxied to ShowBoard by the Home view. Upon receiving the event, ShowBoard will select the best fit model, serialize model parameters and send to the Generator which runs on another Web Worker. The Generator uses SketchRNN from Magenta.js to load the pre-trained models and generate the sample sketches, and then send the sketch back to ShowBoard. ShowBoard will call the renderer to render the sketch on its canvas, offering user the sample sketch.

Benefited from the versatile Vue.js, there's no need to generate dynamic page content in the server-side. Additionally, with the help of webpack, all the pages, scripts, and other files, along with all the dependencies of our system, are packed into a few bundled static assets. What's more, by leveraging TensorFlow.js under Web Workers, we even implement ML, the most resource-consuming part in client side. As a result, the back-end of the system is simply a static web server implemented with python via Flask.

4.3. PROJECT SCOPE

In the context of this project, its scope is limited by the

i) Dataset. There are limited publicly available, complete, and trusted datasets on sketch drawing. Besides, it is extremely hard to build a dataset similar to Quick Draw Dataset by ourselves, because it will be difficult to collect with permission and it will take us a lot of time to review and process the data not only technically but artistically.

ii) Models. Due to the use of the Browser's GPU, and the pursuit of the real-time of detection, it is required to use the small or medium size model which maintains high efficiency and high performance at the same time.

4.4. SYSTEM PERFORMANCE

4.4.1. Classifier performance

We use classification accuracy and loss to measure the performance of our model. The plot of accuracy and loss on training and validation datasets are shown in Figure 7 & 8.

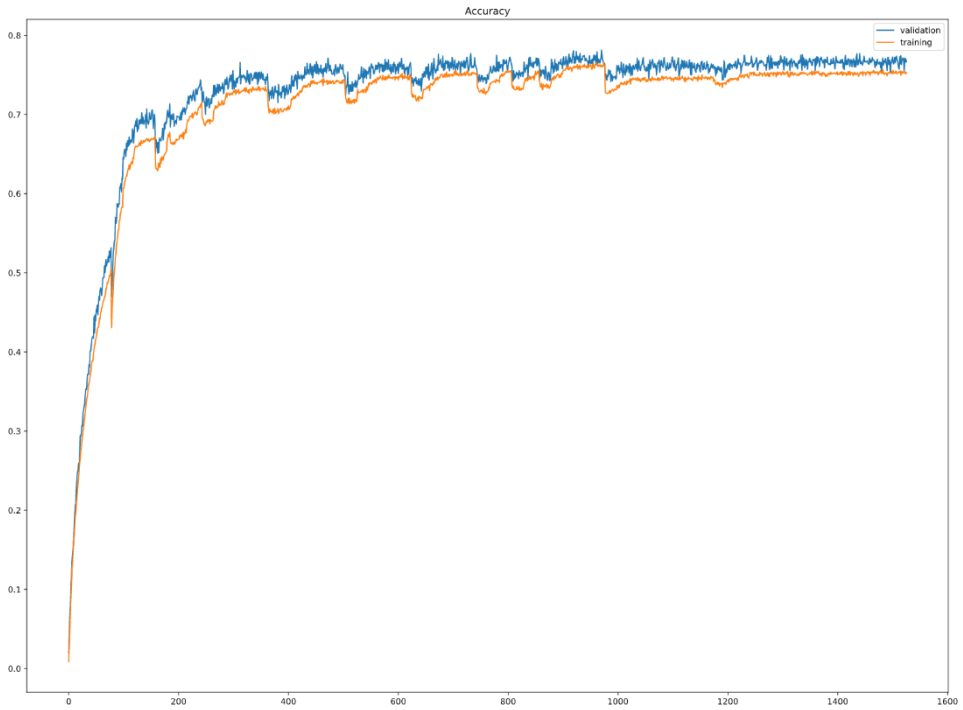

Figure 8: the model accuracy plot on training and validation datasets

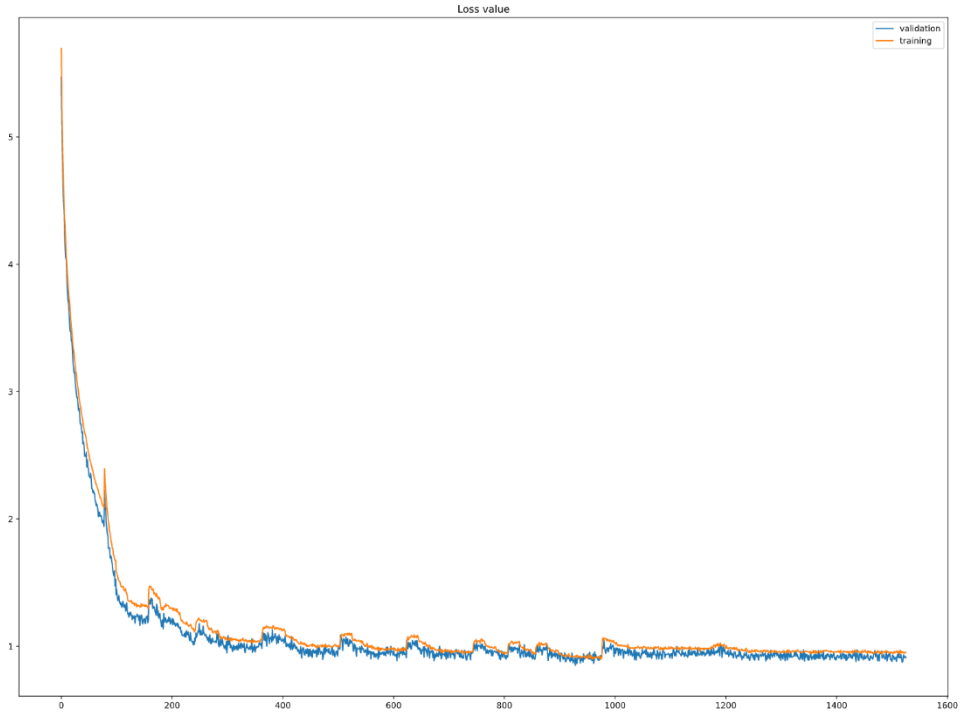

Figure 9: the model loss plot on training and validation datasets

As shown in the plots, the accuracy of our model (mearly 80%) is adequate, however the diverges in the loss plot indicates that the learning rate is too high in the last few hundreds epochs, and we may need a better learning rate scheduler. Moreover, since the loss has nearly no improvement in the last 1000 epochs, our model seems a bit simple for handling the sketch classification.

4.4.2. Generator performance

The performance of generator could be measured by the 'probability' of the model assigns to each sketch, and the output ranking by those probabilities. Multiplying the values for each stroke is one of the natural ways to get the probability for a whole sketch. However, we pick some sketch drawings from the dataset randomly, and we find that the number of strokes of per sketch drawing varies obviously, as shown in Figure 7.

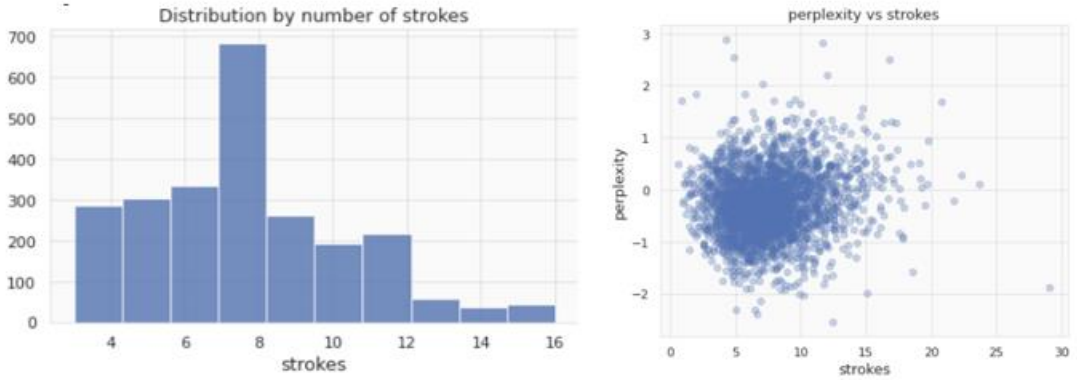

Figure 7: the distribution (left) and perplexity (right) by number of strokes of the "rabbit" category

So we use a reconstruction loss, which is normalized by the number of strokes. We could select from the sketches with the best and worst unnormalized values to show some best and worst samples. Take the "rabbit" category for example, the result is shown in Figure 8.

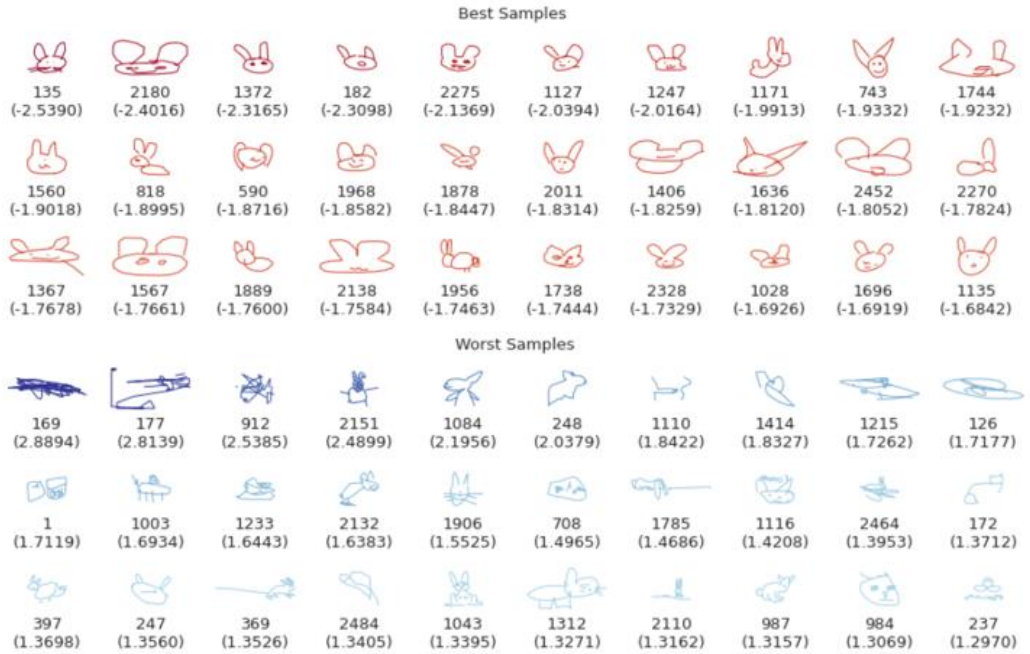

Figure 10: some best and worst samples of the “rabbit” category

4.4.3. General performance

The successful application scenario is shown in the figure. The system favorably identified the user's sketch drawing “a circle” in the left canvas as a hamburger, and generate some samples in the right canvas by machine.

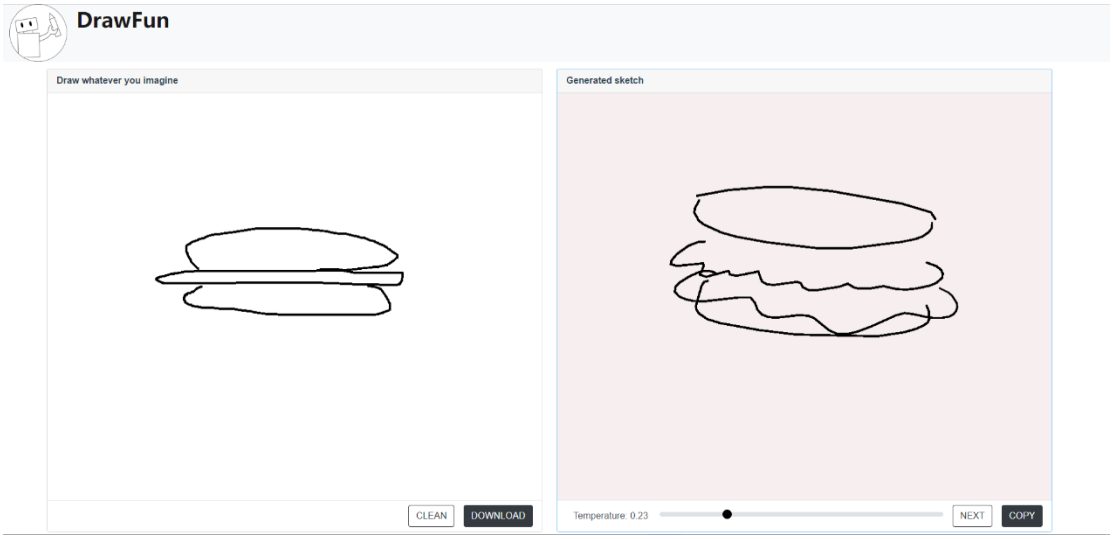

Figure 11: the successful application scenario

5. LIMITATIONS

From the perspective of application, the functions we have realized are relatively limited and cannot meet the various requirements for the complexity, diversity and innovation of artistic creation. In terms of technology, the classifier we have trained is not accurate enough at present, which leads to the incorrect result of detecting the content drawn by users sometimes.

6. CONCLUSION

Our team had a wonderful but also hard time working on this project, and definitely picked up useful skills along the way.

Model construction was a crucial part of the entire process, without the prediction models, we wouldn't have been able to build on this creative system based on simple strokes. Dataset also performed great effect. We can use machine to learn drawing based on the large scales of data. Our group now has a better understanding of how to apply the machine learning method into real project.

Building the system itself presented a whole new set of learning points. We got to apply practical knowledge of machine learning, as well as tap on our existing expertise in Python, JavaScript, and front-end frames like Vue.js and Node.js. Working on the exercise together allowed everyone to learn technical skills from one another, although we were at different space. E-learning happened from in-class to out-class.

Overall, it was truly a multi-dimensional problem due to the separate models design and large-scale dataset. Machine learning on large-scale dataset can create much more creative outputs which can make our life more intelligent and colorful.

5.1 IMPROVEMENTS

If we have a longer time frame to work on this project, we would make the following improvements to our project:

i) Add more basic function of drawing canvas.

For the sake of optimize the user experience during painting, add a variety of brush colors, brush sizes, erasers, background colors and other adjustments to the canvas.

ii) Improve the performance of the classifier.

We supposed that the performance of the current classifier could be roughly improved by two ways. Firstly, the number of hidden units in our LSTM layer seems too small (128*3). As stated in the first part of section 4.4, both training loss and validation loss approximately have no change in relatively early stage of the training process. If we increase the unit size of the first two LSTM layers to 256 (which result in a structure of 256*2, 128), the total parameters will increase from 1,142,495 to 3,077,855, which is still smaller than the total number of our training data (about 3,450,000), and our model will be much complex than now. While the number of parameters in our model increases, it will take even longer time to train our model. As a result, we also plan to try some other RNN variants, like Q-RNN or SRU.

Additionally, since we are doing classification on image data, the spatial information of the sketch is likely missing when trained with RNN. Thus, we proposed two new

model schemes. The first one is a simple concatenation of a CNN subnet and a LSTM subnet, as shown in Figure 11. In the second proposal we will first feed the vector-format data to LSTM (or other RNN variants based on previous experiment), take the full sequence as output. After that, we will rasterize the output sequences (features per point) into 2D matrix with same shapes as the original sketch image and treat each feature as a channel, and then feed the rasterized data to CNN, as shown in Figure 12.

iii) Generate different sketch drawing styles.

One of the limitations from using corporations train models is that the data used is determined by corporations. As a result, we could only generate similar results to the original by using the off-the-shelf model. But in the actual application scenario, each user's artistic aesthetics and pursuit are different. Therefore, training the sketchRNN model with customed datasets of more stick figures with different sketch drawing styles, so as to provide users with more choices, references and inspirations, would be more feasible and practicable.

7. BIBLIOGRAPHY

A Neural Representation of Sketch Drawings
https://arxiv.org/pdf/1704.03477.pdf

Recurrent Neural Networks for Drawing Classification
https://github.com/tensorflow/docs/blob/master/site/en/r1/tutorials/sequences/recurrent_quickdraw.md#recurrent-neural-networks-for-drawing-classification

Magenta SketchRNN
https://github.com/magenta/magenta/tree/master/magenta/models/sketch_rnn

SketchRNN Javascript Implementationon
https://magenta.github.io/magenta-js/sketch/

Tensorflow for JS
https://www.tensorflow.org/js
https://github.com/tensorflow/tfjs

Node.js
https://nodejs.org/en/docs/

Vue.js
https://vuejs.org/v2/guide/

Webpack, Bootstrap
https://v5.getbootstrap.com/docs/5.0/getting-started/webpack/

SketchRNN Performance Evaluation
http://colinmorris.github.io/blog/bad_flamingos