TranSketch Dataset: Learning to Transform Sketches

Luming Chen

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-92
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-92.html

May 29, 2020

Acknowledgement

I would like to thank my research advisor Professor John F. Canny for the support and advice, and Forrest Huang for his awesome and encouraging mentorship. I would also like to thank Professor Björn Hartmann for providing advice on this report. Finally, I want to thank everyone who supported me through this journey.

TranSketch Dataset: Learning to Transform Sketches

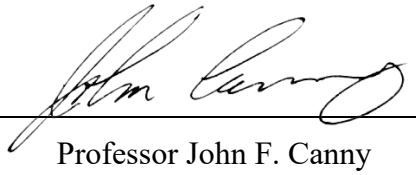
by Luming Chen

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:



Professor John F. Canny
Research Advisor

May 28, 2020

(Date)

* * * * * * *

Professor Björn Hartmann
Second Reader

5/28/2020

(Date)

Abstract

This report introduces a new dataset TranSketch, which is a collection of 20,032 tree sketch pairs, each accompanied with a natural language description describing the instructions needed to transform one sketch to the other. To provide more options for exploration, for each sketch, we also include a latent representation produced by the encoder of a Sketch-RNN model. The TranSketch dataset provides fine-grain transformation between stroke-based sketch pairs using stylistic content text descriptions. Our statistical analysis suggests tree component-level transformation to be the most promising direction for investigation, but there are certainly more to be explored for this dataset.

Contents

1. Introduction

Sketching is a natural way for people to express artistic feelings and convey abstract information. In recent years, sketch-related deep-learning research, such as sketch generation, has gained popularity. The interaction between text and sketch is also increasingly drawing people's attention. Sketch datasets such as SketchyScene[7] and CoDraw[3] were widely used in those researches. However, if we want to specifically explore sketch transformation based on stylistic content text descriptions, such datasets are not enough.

This report presents a new dataset TranSketch to address the challenge. TranSketch provides a fine-grain transformation between stroke-based sketches using natural language. TranSketch contains 20,032 responses collected from Amazon Mechanical Turk. Each response in the dataset consists of an original sketch, a latent representation of the original sketch, a target sketch, a latent representation of the target sketch, and a user-provided description that describes the instructions needed to transform the original sketch to the target sketch. The latent representation is a latent vector generated by the encoder of a Sketch-RNN model, which is a Sequence-to-Sequence Variational Autoencoder (VAE). All sketches are from the Quick, Draw! dataset[1]. Currently, our dataset only contains sketches from the tree category, but the same data collection process can be performed for other categories.

Our dataset will allow people to explore and develop applications related to language descriptions that correlate and translate between sketches. Some examples include transforming sketches based on language input or predicting transformation instructions based on two given sketches.

2. Related Work

Datasets have played a critical role throughout the evolution of sketch-related machine-learning research. They not only provide means for training and evaluating models, but also inspire researchers to explore in more creative and challenging directions. This section attempts to address a few datasets targeting three major sketch-related research directions, which are sketch labeling, sketch-based image retrieval, and collaborative sketching.

2.1 Sketch labeling

The task of sketch labeling requires a sketch to be labeled to a category, such as tree, key, airplane, and so on, or to a captioning that describes at the object or scene level. The Quick, Draw![2] dataset and the SketchyScene[7] dataset are examples of sketch categorization and sketch captioning respectively.

The Quick, Draw! dataset contains a collection of 50 million drawings, divided into 345 categories. All sketches were created by human users in a game in 20 seconds. In contrast with most of the existing image datasets that store images as pixels, the Quick, Draw! Dataset represents a sketch as a set of pen stroke actions. Its great amount of doodling data provides many opportunities for researchers for developing and studying sketch-related machine learning techniques.

SketchyScene is a large-scale dataset consisting of both object- and scene-level data with rich annotations. It contains more than 29,000 scene-level sketches, 7,000 pairs of scene templates and photos, and 11,000 object sketches, accompanied by ground-truth semantic and instance masks. The highly scalable and extensible nature of the dataset allows researchers to investigate in the direction of scene composition and semantic segmentation of scene sketches.

2.2 Sketch-based Image Retrieval

Sketch-based image retrieval is a frequently studied computer vision problem with a long history. The basic task usually involves users providing a simple sketch with minimal user-defined features and asking the computer to return similar images. Datasets such as the Sketchy database[6] are developed primarily to assist research in fine-grained image retrieval.

The Sketchy database contains a large collection of sketch and image pairs from 125 categories. Crowd workers were asked to sketch according to particular photographic objects, and 75,471 sketches were produced out of 12,500 objects. The Sketchy dataset provides a fine-grained association between the sketch details and photographic object details. The nature of the dataset allows training for cross-domain representation so that the image retrieval task can go beyond category recognition and instead focus on smaller features such as pose, parts, and sub-type.

2.3 Mixed Initiative Sketching

The task of mixed initiative sketching focuses on building AI agents or ML systems that can assist humans in editing and completing a sketch. Datasets addressing such a task must contain means of human-computer interaction, which are usually natural language descriptions, and the progression of the sketch under the means. The CoDraw dataset[3] is one of the examples.

The CoDraw dataset was collected from a game involving two players: a Teller and a Drawer. In the beginning, the Teller is given an abstract scene with multiple clip art pieces while the Drawer is given a blank canvas with available clip art pieces. The goal of the game is to let the Drawer reconstruct the scene solely based on the communication with the Teller. Teller was given the chance to 'peek' once at Drawer's canvas during each game. The CoDraw dataset consists of 9993 sessions of conversation records in natural language, scene modifications, and ground-truth scenes.

Our dataset falls under the category of mixed initiative sketching. The main problem we address is the transformation of stroke-based sketches based on

stylistic language descriptions. However, our dataset also opens up new opportunities for sketch and natural language understanding and synthesis.

3. Data Collection

3.1 Sketch

All sketches in our dataset come from the tree category of the Quick, Draw! Dataset. The tree category contains 70k training samples, 2.5k validation, and 2.5k test samples. We used the 70k training samples to form 35k pairs, displayed around 21k for text collection, and eventually collected 20032 responses. Therefore, our dataset contains 20032 sketch pairs, and every sketch is unique.

We represent every sketch as a set of pen stroke actions, and each sketch stroke was converted to and stored with the Stroke-5 format $(\Delta x,\ \Delta y,\ p_1, p_2, p_3)$. Δx and Δy are the offset distance in the x and y directions of the pen from the previous point. p_1, p_2, and p_3 represent a binary one-hot vector of 3 possible pen states. If p_1 is set to 1, indicating the pen is touching the paper, and a line will be drawn from the current point to the next point. If p_2 is set to 1, the pen is lifted from the paper, and no line will be drawn from the current point to the next point. Finally, if p_3 is set to 1, the drawing is finished, and no future points, including the current one, will be drawn.

3.2 Text descriptions

We created an application to conduct sketch comparisons and collect text descriptions. In each comparison, we displayed a randomly sampled tree sketch pair to a participant. All sketches were scaled individually to have relatively the same size during display. Then we asked the participant to give instructions needed to transform the sketch on the left to the sketch on the right in English (eg. "Make the trunk of the tree taller and leaves

rounder"). The application was put on Amazon Mechanical Turk to collect crowdsourced data.

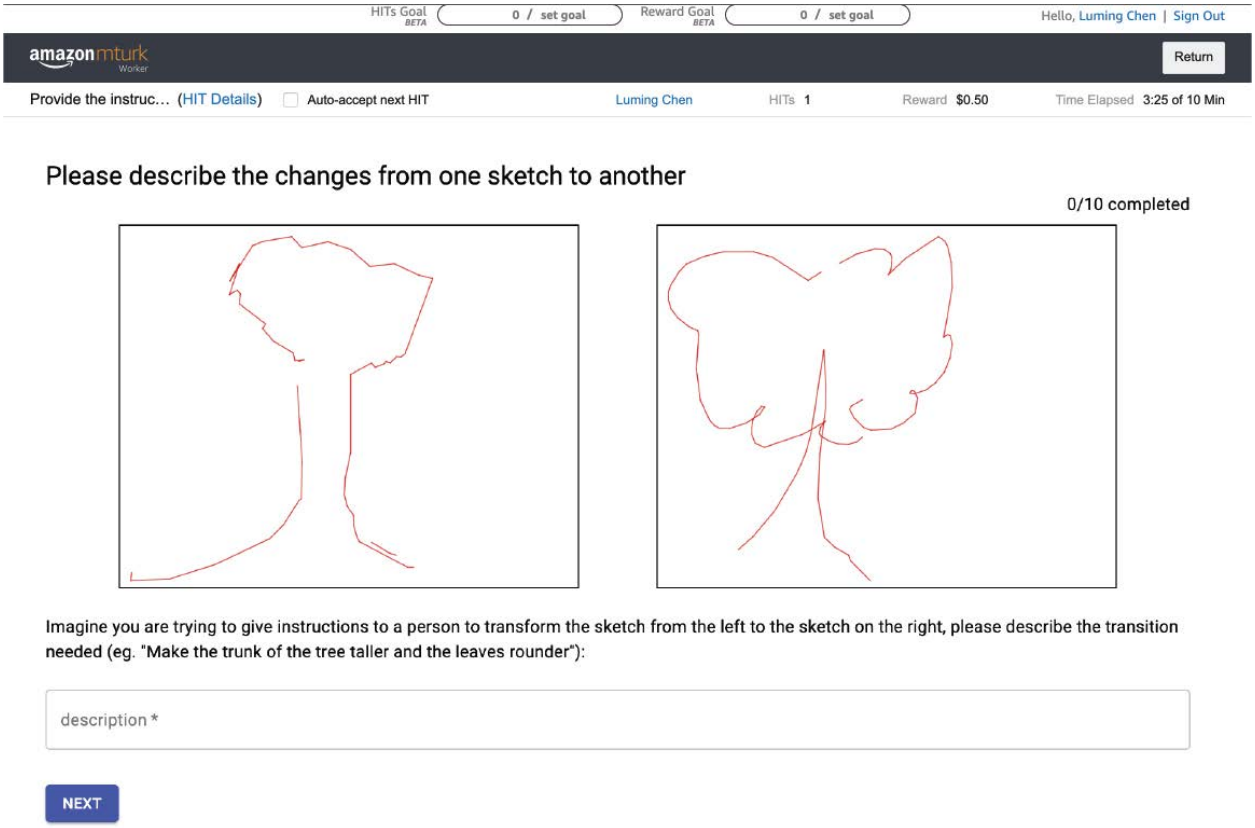


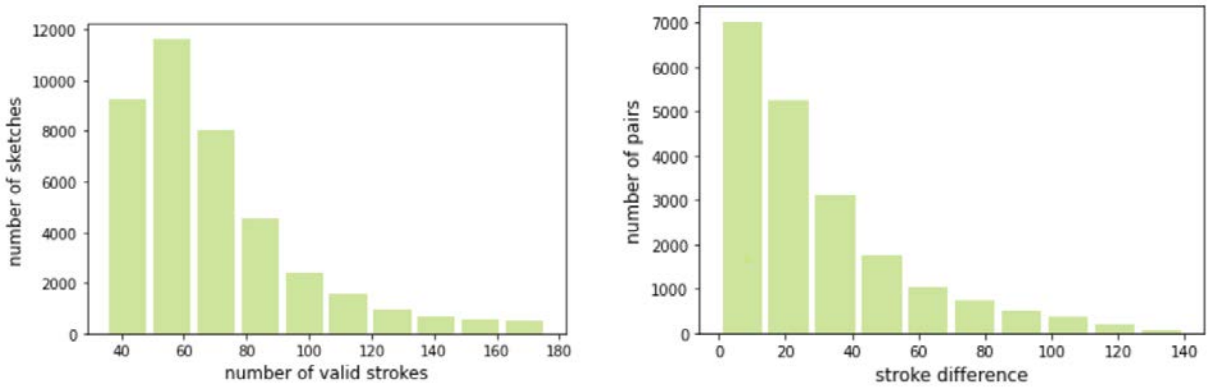
Figure 3.1: Our data collection interface on Amazon Mechanical Turk: At each turn, two tree sketches are displayed, and participants are required to describe the changes required to convert the sketch on the left to the sketch on the right.

4. Dataset Statistics and Analysis

In this section, we provide statistical insights and analysis for our dataset. We hope the statistics and analysis from different perspectives can shed some light on the potential problems to be explored.

4.1 Sketch Statistics

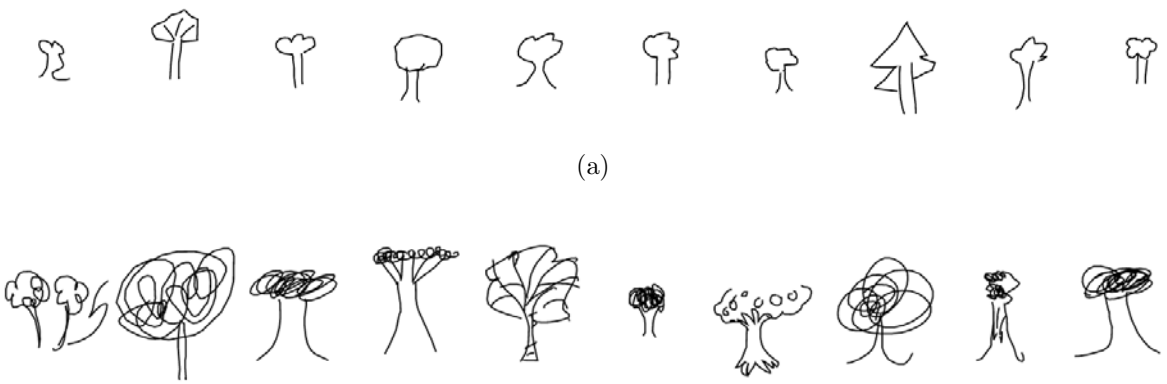
Our dataset contains 40064 sketches. For every stroke, the first two elements are the offset distance from the previous point. The offset values are generally very small, with an average of 0.0012, and a standard deviation of 0.624.



(a) (b)

Figure 4.1: (a) The distribution of the number of strokes for all sketches. (b) The distribution of the stroke difference between the original sketch and the target sketch in each pair.

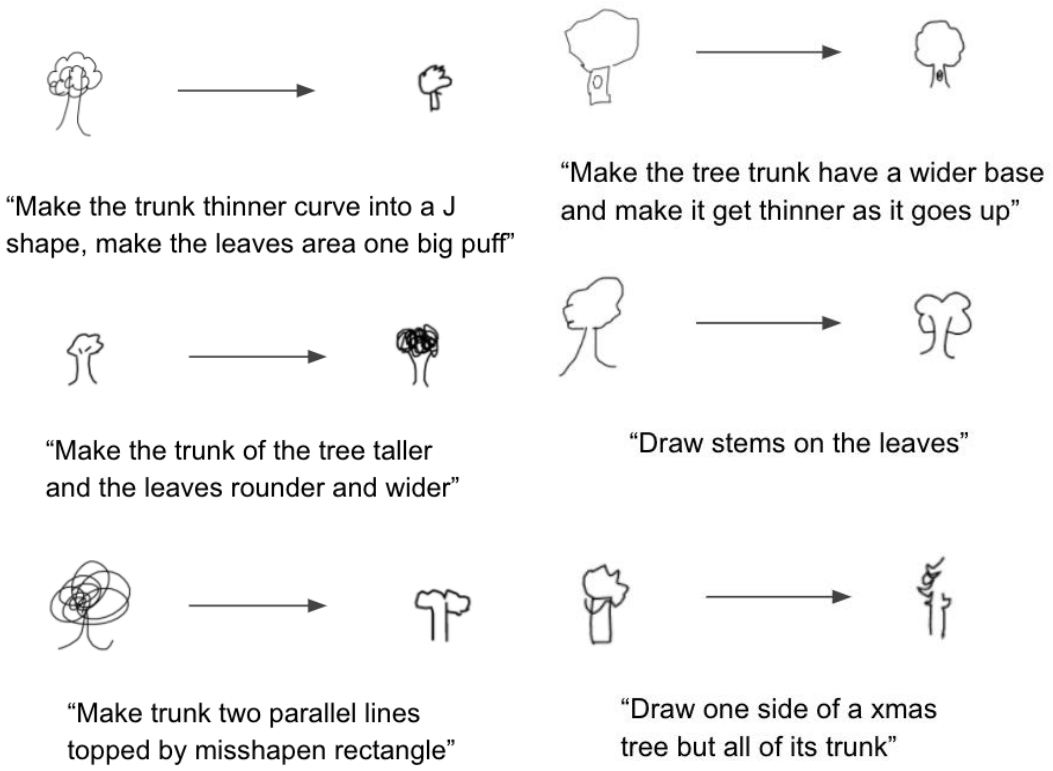
Sketches in our dataset have various strokes, but to make the format uniform, sketches were padded to the length of 176 with the padding stroke [0, 0, 0,



(a)

(b)

Figure 4.2: (a) 10 sketches with 35 strokes. (b) 10 sketches with 176 strokes.



(a)

(b)

Figure 4.3: (a) Sketch pairs with the most stroke differences. (b) Sketch pairs with the least stroke differences.

0, 1]. We plot the number of strokes for all sketches in Figure 4.1(a) and show the distribution. The average strokes for a sketch is 69.48 strokes, with a minimum of 35 and a maximum of 176 strokes. Most of the sketches have

strokes between 35 and 80. Figure 4.2 (a) draws out 10 sketches with 35 strokes while Figure 4.2 (b) with 176 strokes. The number of strokes do not affect the quality of a sketch, but sketches with more strokes generally look more complex, detailed, and messy. We plot out the stroke difference between the original sketch and the target sketch in each pair in Figure 4.1(b). The distribution is skewed to the right. Most of the pairs have relatively similar numbers of strokes. However, as shown in Figure 4.3, larger stroke number differences do not indicate more differences between the sketches nor more differences between text descriptions.

4.2 Latent Vector Statistics

To provide people the option to explore sketches in the latent space, every sketch in the dataset has a latent representation. The latent vectors were generated using the Sketch-RNN model proposed in the paper *A Neural Representation of Sketch Drawing*[1]. Sketch-RNN is a recurrent neural network (RNN) that can construct stroke-based drawings of common objects. Its architecture is a Sequence-to-Sequence Variational Autoencoder (VAE). We trained a Sketch-RNN model on the tree category of the Quick, Draw! dataset. Then, we fed each sketch to the encoder to obtain its latent representation.

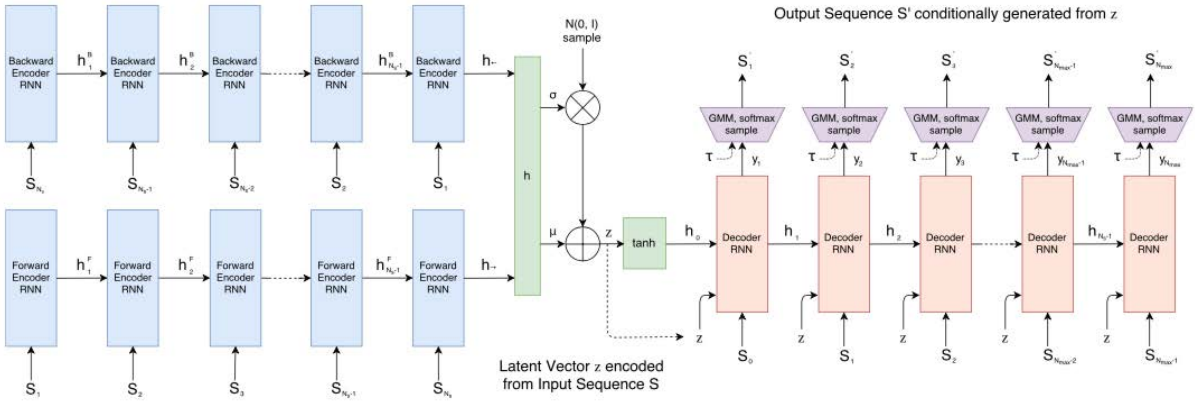
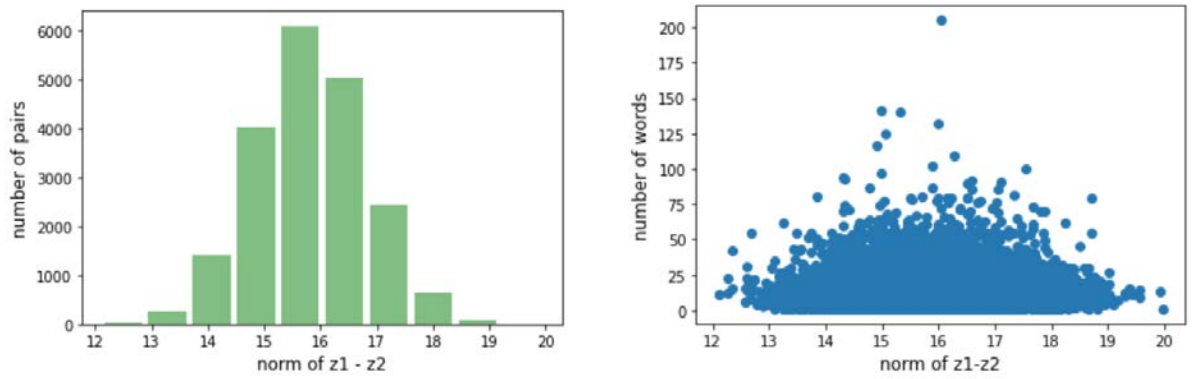


Figure 4.4: Schematic diagram of Sketch-Rnn [1]

Every latent vector has 128 dimensions. All latent vector elements have an average value of 0.00099 and a standard deviation of 0.993. In Figure 4.5(a), we plot out the distance in latent space (z1 - z2) between the latent vector



(a) (b)

Figure 4.5: (a) The distribution of the distance between the latent vector of the original sketch z1 and the latent vector of the target sketch z2 in each pair. (b) The distance in latent space vs. the number of words in text description.

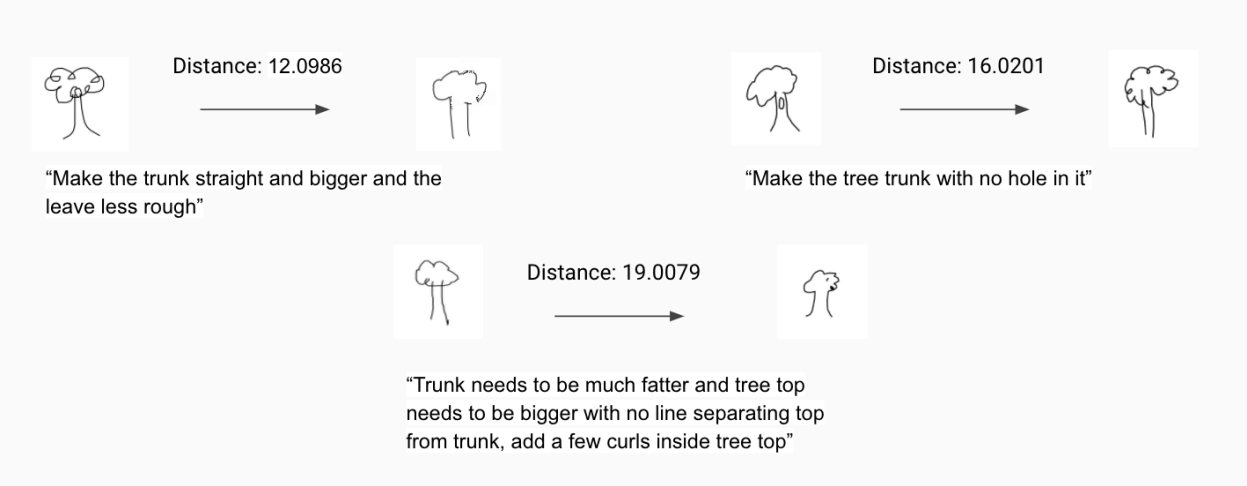


Figure 4.6: Sketch pairs with different latent distances

of the original sketch z1 and the latent vector of the target sketch z2. The distribution of the distance follows the shape of a Guassian distribution, and most pairs have a distance between 14.5 - 17.5. As shown in Figure 4.6, a bigger distance in latent space does not imply a larger difference in sketch. Figure 4.5(b) plots out the relationship between the distance in latent space and the number of word in text description. The graph has an interesting bell shape, which can be further investigated. Distance in latent space may correspond to other interesting details, which are left for researchers to explore in the future.

4.3 Text Description Statistics

Text description is the main component that differentiates our datasets from other common sketch datasets. Each text description provides the instructions to transform an original sketch to a target sketch. Many of the descriptions tend to focus on part-level transformation, such as "Make the trunk thinner" and "Make the leaves smaller". They usually encompass the most salient parts of the change, like "make the truck more of a triangle and remove swirls from leaves", but some also provide detailed instructions for drawing, as in "Draw a line from the leafs to the top of the page and then mark down and to the left for an 1/8 of an inch".

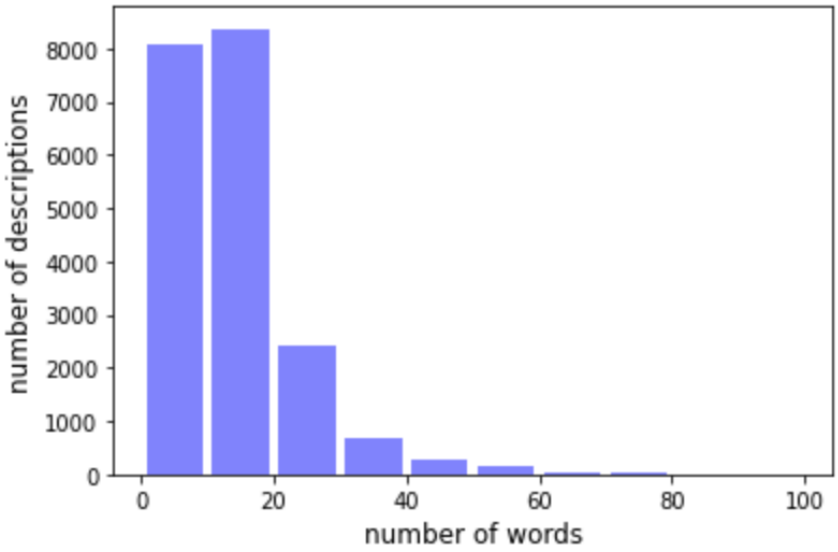
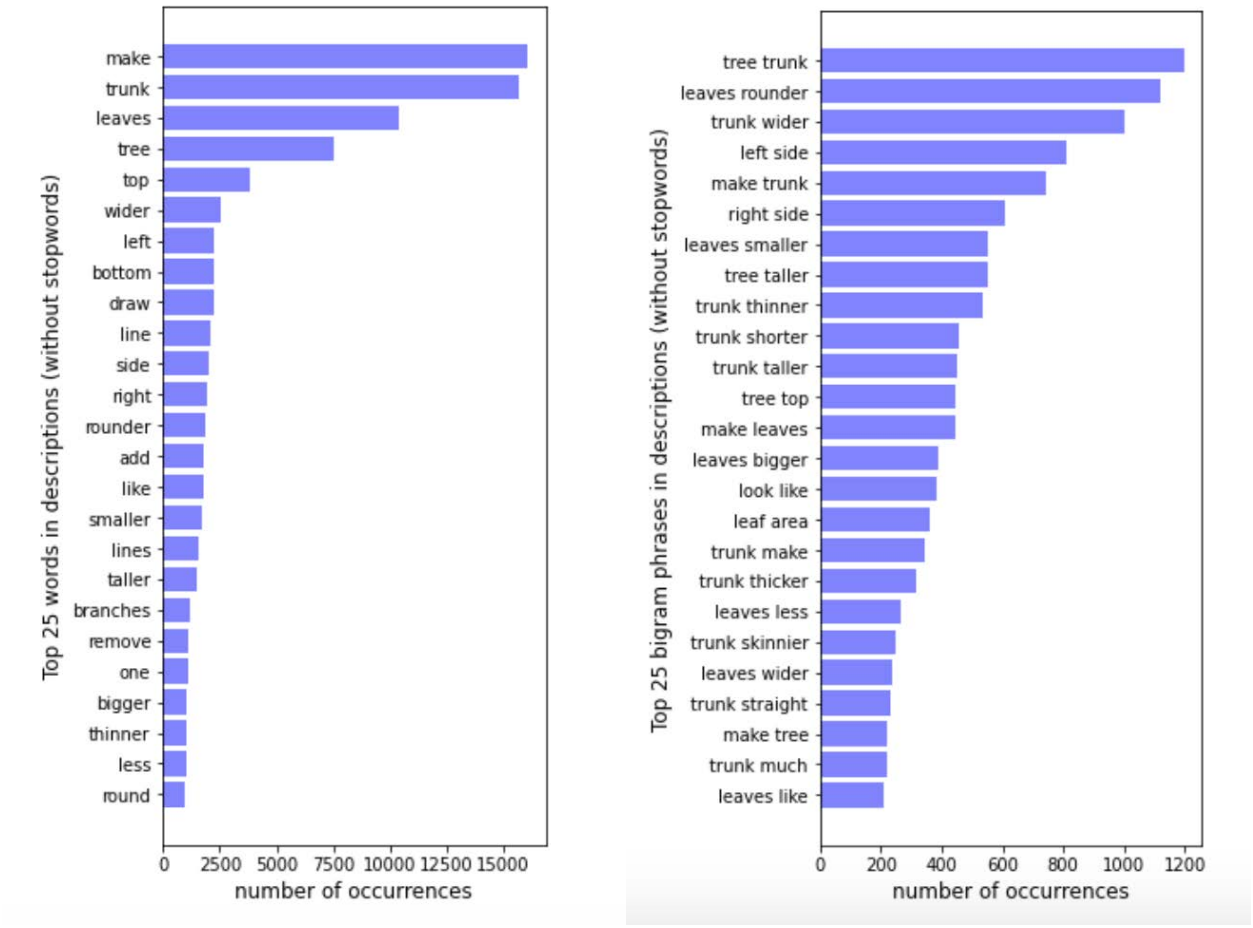


Figure 4.7: The distribution of the length (word counts) of all descriptions

Figure 4.7 shows the distribution of the length (word counts) of all descriptions, excluding 8 outliers that have more than 100 words. The average word count for a description is 13.29 words, with a minimum of 1 and a maximum of 205 words. Most of the descriptions have a length between 1 and 20 words. In Figure 4.8(a), we plot the most common words in all descriptions, excluding all stopwords. Tree components such as "trunk", "leaves", "top", "bottom" and "branches" and transformative words such as "wider", "rounder", "smaller" and "taller" occur very frequently. "Trunk" seems to be the most popular tree component people paid attention to. We also plot out the most common bigram phrases in Figure 4.8(b). Again, we excluded all



(a) (b)

Figure 4.8: (a) A plot of the most common words in text descriptions, excluding all stopwords. (b) A plot of the most common bigram phrases, excluding all stopwords.

stopwords. Phases that describe component change such as "leaves rounder", "leaves smaller", "trunk taller", and "trunk thinner" are most frequently used. Phases that target a tree part like "tree trunk", "tree top" and "leaf area" are often mentioned as well.

4.4 Noise

Given that we could not easily monitor the user inputs on Amazon Mechanical Turk, the text description data may contain a few invalid answers. We strongly suggest applying filters to the responses before doing any training. The most common invalid user inputs we encountered were "left", "right",

"tree", "none", "neutral", "good" and "bad". Thus, people should be cautious about the very short answers. Some of the "Make the trunk of the tree taller and the leaves rounder", as we observed, are also invalid as this sentence is used as an example for user input. For each assignment on AMT, a worker was asked to complete 10 comparisons. Therefore, if the same phrase appears more than 10 times, people may also want to pay attention to its validity.

Compared to text descriptions, sketches should be more clean as they were collected from the Quick, Draw! Dataset. We eyeballed all the sketches and filtered out inappropriate and ugly ones before showing on AMT. However, we may still miss some low-quality ones.

5. Conclusion and Discussion

We introduce a new dataset for stroke-based sketch transformation based on stylistic content text description. With over 20k responses, the dataset should provide rich contextual information for sketch transformation. Data statistics suggests tree component-level transformation may be the most promising direction to be explored while line-level transformation may also be worth a try.

There are also several other potential directions on our datasets. Instead of generating new sketches based on some original sketches and instructions, we can investigate the prediction of instructions based on sketches. Besides, we can collect data for other categories utilizing the same collection procedure and explore cross-category transformation. As suggested in A Neural Representation of Sketch Drawing, by interpolating between latent vectors from two different categories, one sketch can be morphed into another one. Similar attempt can be used for our dataset.

After the bibliography, we provide a report of a class project, which is an explorative attempt on building models that transform sketches based on text descriptions using this dataset. It shows the potential of the dataset and what it can achieve.

Bibliography

[1] David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

[2] Jonas Jongejan, Henry Rowley, Takashi Kawashima, Jongmin Kim, and Nick Fox-Gieg. The quick, draw!-ai experiment. *Mount View, CA, accessed Feb*, 17:2018, 2016.

[3] Jin-Hwa Kim, Nikita Kitaev, Xinlei Chen, Marcus Rohrbach, Yuandong Tian, Dhruv Batra, and Devi Parikh. CoDraw: Collaborative Drawing as a Testbed for Grounded Goal-driven Communication. *arXiv preprint arXiv:1712.05558*, 2019.

[4] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalanditis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. 2016.

[5] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[6] Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Transactions on Graphics (proceedings of SIGGRAPH)*, 2016.

[7] Changqing Zou, Qian Yu, Ruofei Du, Haoran Mo, Yi-Zhe Song, Tao Xiang, Chengying Gao, Baoquan Chen, and Hao Zhang. Sketchyscene: Richly-annotated scene sketches. In *ECCV*, pages 438–454. Springer International Publishing, 2018.

A Neural Transformation of Sketch Drawings

Luming Chen, Zhi Chen, Zangnan Yu, Xinyun Zhang
UC Berkeley
{lumingc, zhichen98, yuzan, lily1128}@berkeley.edu

ABSTRACT
In our work, we addressed a completely new task that combines generative modeling of sketches and the interaction between text and sketches. We intend to explore language descriptions that correlate and translate between sketches and build an ML system that transforms sketched objects based on stylistic and content text descriptions, such as 'Make the branch of a tree taller'. We developed four methods with different neural network architectures: latent vector transformation, sketch to sketch transformation using CNN encoder - CNN decoder, CNN encoder - RNN decoder, and RNN encoder - RNN decoder. With all models manually evaluated by volunteers and an inception model, we conclude the second version of RNN encoder - RNN decoder achieves the best result.

INTRODUCTION
In recent years, generation of sketch-based drawing has increasingly drawn people's attention. People proposed different generative modeling techniques, such as Variational Autoencoder (VAE), Generative Adversarial Network (GAN), and Autoregressive (AR) models to generate images. More specifically, in [4], the authors introduced Sketch-RNN, a model that is tailored to generating sketches of common objects.

The interaction between text and images have been studied extensively. Image captioning, which is the process of generating textual description of images, has been studied for a long time, and the state-of-the-art model uses CNN to extract features from image and RNN to generate the description. The reverse of image captioning, which is the process of generating images from natural language descriptions, has also been studied in recent years. In [10], the authors proposed a model to generate images conditioned on captions.

One question still remains: Can we teach machines to transform sketches like humans? In our research, the main task we are exploring is that given a human drawn sketch and a short piece of text, how to let machines transform the sketch to another sketch based on the textual description. For instance, if we are given a short tree and a piece of text saying "grow taller", we want to transform the tree to a taller one. We developed four different methods using neural networks: latent vector transformation, sketch to sketch transformation using CNN encoder - CNN decoder, CNN encoder - RNN decoder, and RNN encoder - RNN decoder.

RELATED WORK
In the paper *A Neural Representation of Sketch Drawing*, Ha and Eck proposed Sketch-RNN [4] which primarily focused on constructing stroke based drawings of common objects. Unlike other previous works that learns from sketch images [9] or uses sampling to generate blurry pixels [2], Sketch-RNN is a neural network based autoencoder that generates stroke sequences. The quality of Sketch-RNN generation inspired us to look into both its latent space and model structure.

Based on findings from Sketch-RNN, researchers have explored sketch generation with extra human inputs as in Sketchforme [7] and sketch translation into image outputs as in SketchyGAN [3]. These recent successes show that models are capable of both decoding sketch strokes from data beyond the stroke sequences, and encoding sketches to inform the generation of other types of data. In our attempts, we experimented with text sequences, image pixels, in addition to sketch strokes in different combinations to achieve the goal of sketch transformation with human inputs.

Previous works have proposed learning user-specified image transformation from unlabeled data to assist data augmentation process [11], as well as music sequence transformation with generative models [1]. In comparison, our approach aims at transforming the semantic content to generate new sketches instead of altering the style or local features of the data.

Although our work only used one category of sketch from Sketch-RNN *Quick, Draw!* dataset [8], the models and methods we developed can be directly transferred to data from any other categories in this dataset. To better format and visualize the experiment results, we also borrowed several helper functions from the Magenta Project [4] which was originally developed for Sketch-RNN.

DATASET
Data Collection
We created an application to conduct sketch comparison tasks. In each comparison, we displayed a randomly sampled tree sketch pair to a participant. All sketches were scaled individually to have relatively the same size during display. Then we asked the participant to give instructions needed to transform the sketch on the left to the sketch on the right in English (eg. "Make the trunk of the tree taller and leaves rounder"). We put the application on Amazon Mechanical Turk and collected 20032 responses in total.

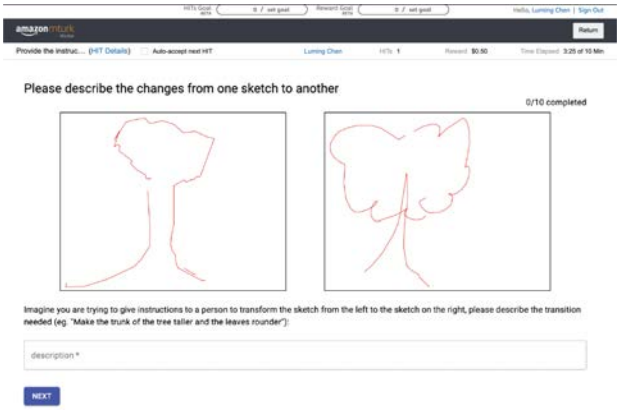


Figure 1: Our data collection interface on Amazon Mechanical Turk: At each turn, two tree sketches are displayed, and participants are required to describe the changes required to convert the sketch on the left to the sketch on the right.

All sketches displayed in the task are from the tree category of the *Quick, Draw!* dataset. The sketches in *Quick, Draw!* were created by human users in a game in 20 seconds. Each sketch stroke was converted to the Stroke-5 format (Δx, Δy, p_1,p_2,p_3) for both displaying and training models. Δx and Δy are the offset distance in the x and y directions of the pen from the previous point. p_1,p_2, and p_3 represent a binary one-hot vector of 3 possible pen states. If p_1 is set to 1, it indicates the pen is touching the paper, and a line will be drawn from the current point to the next point. If p_2 is set to 1, the pen is lifted from the paper, and no line will be drawn from the current point to the next point. Finally, if p_3 is set to 1, the drawing is finished, and no future points, including the current one, will be drawn.

Data Filtering

Given that we could not easily monitor the user inputs on Amazon Mechanical Turk, we applied a strict and robust filtering for data quality assurance. We first filtered out all user responses that are less than 6 characters. Then, we found out all responses that repeated more than 9 times since each task on Amazon Mechanical Turk contained 10 comparisons. Among all the highly repeated responses, we manually check the validity of each one and white-listed all valid phases. Lastly, we filtered out all responses that have more than 6 repeated words. After filtering, 14413 responses remained. We used 80% of the remaining responses for training, 10% for validation, and 10% for testing.

METHODOLOGY

Latent Vector Transformation

As previous works[4] have presented the Sketch-RNN, a neural network that is able to construct stroke-based drawings, we first think about how to make use of the Sketch-RNN to generate transformed sketches. Sketch-RNN will transform a series of strokes into latent vectors with its encoder and will output a sketch by decoder with latent vector. We would like to build on top of the Sketch-RNN model by transforming the latent vectors. Here we explore in the latent space, not only because the latent vector is short, which will lead to a faster computation process, but also because there will be less noise in the latent transformation process.

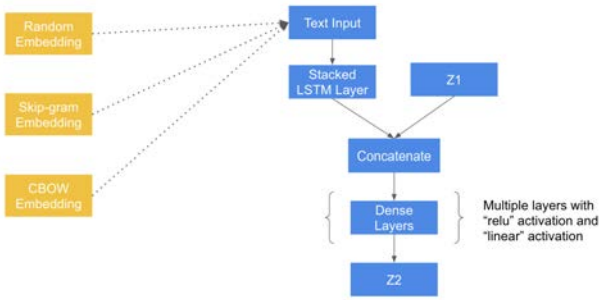


Figure 2: Schematic diagram of Vector Transformation model

There are two inputs for our model: original sketch and description text. For the original sketch, in this model, we used the latent vector z from the Sketch-RNN model. For the text, we have tried different embeddings including Bert embedding, skip-gram embedding and CBOW embedding which can transform sentences into vectors.

Next, we put an embedded vector into an LSTM layer in order to find sequential relations in the text. Since latent vector z did not seem to have sequential relationship among its elements, we decided not to use an RNN layer, and instead directly concatenate the output of the LSTM layer and z-vector for the original sketch together. Then we use several dense layers with relu and linear as activation functions to output the z-vector for the target sketch.

Sketch Transformation

In addition to learning the transformation between latent vectors, we also explored possibilities of learning end-to-end transformations between sketch strokes. To incorporate both text sequence and sketch data in this task, we experimented with three different types of encoder-decoder architectures that all consist of one encoder for the input text instruction, a second encoder for the input sketch, and one decoder to generate the transformed sketch. We used CBOW text embedding for all encoder-decoder models.

1. **CNN Encoder with CNN Decoder** When designing CNN models, we spent a long time solving the problem: given 2D images and 1D text, how to concatenate images with text? After careful consideration, we proposed two major model structures to solve this problem: one with a dense layer, and one without a dense layer.

   Both models use three convolutional 2D layers as encoders. In both models, we first process the text using an Embedding layer and an LSTM layer with 100 output units. The model without a dense layer concatenates the text with the image by reshaping the text to a squared size. In the model, we take the original sketch as input, feed it consecutively into two convolutional 2D layers. Then, we reshape the hidden layers of text output from LSTM into a shape of 10

* 10 tensor, and concatenate with the image output from CNN encoder. Next, we use a linear activation layer. After that, we feed the output after activation into CNN decoder. The decoder consists of three convolutional 2D transpose layers. After feeding the data into the decoder, we use a sigmoid activation layer to get the final output. The model

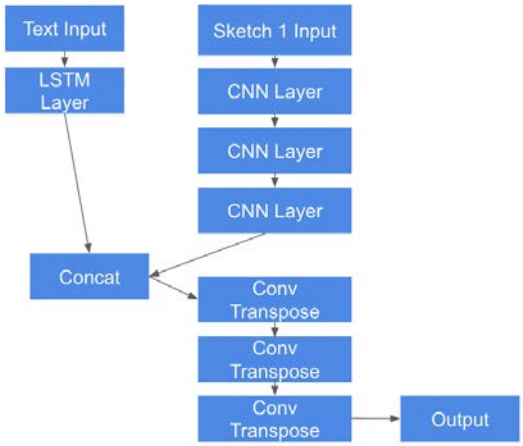


Figure 3: Schematic diagram of CNN Encoder with CNN Decoder model

with a dense layer concatenates the text with the image by concatenating the 1D dense layer with the text. The encoder consists of three convolutional 2D layers. After we got the encoded data, we flatten it from 2D to 1D, and feed it into a Dense layer of 1400 output units. Then, we concatenate the output from the Dense layer with the text output from LSTM. Same as the previous model structure, we also use a linear activation layer, a decoder, which consists of three convolutional 2D transpose layers, and a sigmoid activation layer to get the final output.

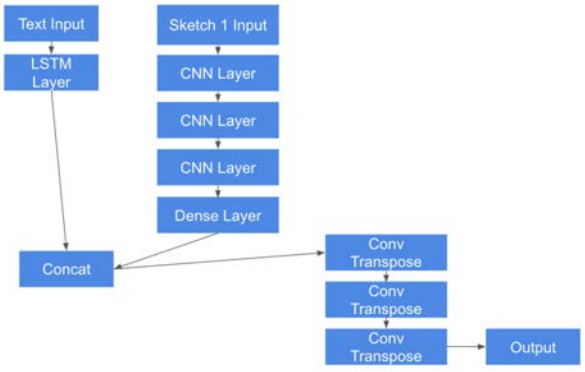


Figure 4: Schematic diagram of CNN Encoder with CNN Decoder model with Dense layer

2. **RNN Encoder with RNN Decoder** In this model, our encoder is two LSTM layers. One LSTM layer takes in the embedded text as input and outputs hidden state h_t and c_t. The other LSTM layer takes in the original sketch as input and outputs hidden state h_s and c_s. Then, we concatenated the two hidden states to get the final encoder states.

Our decoder is an LSTM layer that takes in the encoder states as initial states and the target sketch as input. At each time step i of the decoder LSTM, we feed the previous stroke Si-1 of the target sketch, where S0 is defined as a starting stroke (0, 0, 0, 1, 0). The output of the LSTM layer is fed in a dense layer and then gets split into position and pen states. For the position output, we used a linear activation and used mean square error as our loss. For the pen states output, we used softmax as activation and cross-entropy as loss.

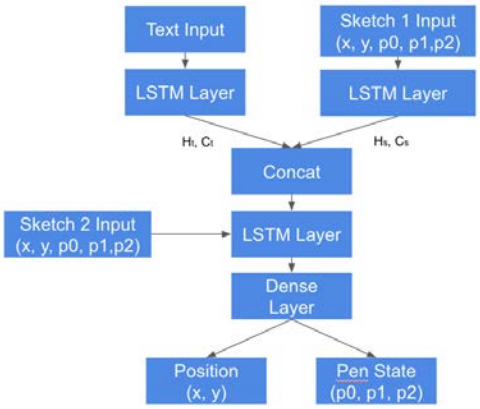


Figure 5: Schematic diagram of RNN Encoder with RNN Decoder model

To decode the sketch, we developed two versions of inferencing. The first version iteratively predicts the next stroke from the previous stroke at each time step, and the second version predicts the whole new sketch in a single time step. For both versions, we first encode the input text and the original sketch to retrieve the initial decoder state. Then, for the first version, we run one step of the decoder with the initial state and the first stroke of the original sketch. The outputted states and stroke will be the input for predicting the next stroke. This stroke prediction will continue until we reach the target length for the predicted sketch. We also tried to feed the starting stroke (0, 0, 0, 1, 0) to the decoder as an initial input, but got very similar outputs for different sketches, possibly due to the simple model architecture. For the second version, we directly feed all strokes of the original sketch and the encoder state as inputs to the decoder. The decoder is run only once for the prediction.

3. **CNN Encoder with RNN Decoder** The text encoding process of this model is the same as what we have tested with all other models introduced in this section. It takes in preprocessed text data and encodes it with one layer of LSTM. The sketch encoding process of this model relies on convolutional neural net instead of recurrent neural net used in the original Sketch-RNN architecture. And by using the CNN encoder, this model takes input as pixel values of the sketch instead of its actual strokes. A potential advantage of CNN in this task is its focus on exploring pixel locality and generating features at

different scales. Because many text inputs in our training data would describe the shape or local feature of a sketch, using a CNN might help to address these instructions. Since we have limited time and computing resources during this work, we built the CNN in this model using many modules of depth-wise convolution followed by 1x1 convolution and batch norm, similar to the structure of MobileNet [6], and added a few max pooling layers in between.

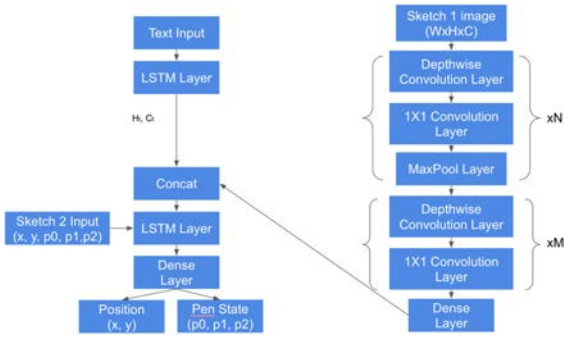


Figure 6: Schematic diagram of CNN Encoder with RNN Decoder model

After these encoding processes, we flatten the encoded sketch image and concatenate it with two hidden layers from the LSTM text encoder. These concatenated results will then go through another fully connected layer to match the shape of hidden states in the LSTM decoder for sketch stroke generation. We have also tested with several dropout layers after different convolutional and/or fully connected layers to prevent over-fitting during the training, but the results were not as good as data augmentation alone. So dropout layers are not included in this model. The LSTM decoder in this model is the same as the first version described in the previous model.

MODEL TRAINING

The encoder-decoder models are not very easy to optimize, so we have made a few adjustments and customization during the training process. By default, the sketch data has a fixed sequence length of 176 strokes. However, most of the sketches in our dataset end within 100 strokes with the remaining strokes being dummy data (zero paddings). Thus we customized the loss function with proper masking to ignore these paddings in the stroke sequence and only computed loss for the valid strokes. Since the first two numbers in any sketch stroke are the offset distances in x and y, we measure their loss with mean square error. On the other hand, the last 3 numbers in a sketch stroke are the pen states, which is represented by a one-hot vector, so we applied categorical cross-entropy as the loss.

For the special model when we use CNN as the decoder to generate sketch images, we simplified the training process into multiple binary classification problems. The sketches in our dataset don't have any color or transparency, so it's safe to predict 1 or 0 for each pixel of the generated sketch images. Hence we used binary cross-entropy loss for this task.

Another adjustment for models with CNN encoder is the preprocessing of sketch data. This sketch dataset does not specify either canvas size or color map for people to display each sketch as an image. We customized some sketch drawing tools from the Magenta Project [4] to draw each sketch into a squared PNG image and scale them to the same size. During this scaling process, it is important for us to keep the aspect ratio of each sketch, because our sketch transformation task involves stretching the tree to be relatively taller or wider based on the text instructions. We also generated these sketch images with white strokes and black background, hoping to make it easier for the convolutional neural net to pick up the useful signals.

EXPERIMENTS AND RESULTS

We used two exploratory ways to evaluate our generative models. We conducted user studies to rate our model outputs and developed an inception model to classify the predictions. The latent vector transformation model was not used for evaluation because the outputs of this model have high similarities regardless of inputs, which implies the model did not learn enough to distinguish different cases, possibly due to the very simple architecture. Different embeddings also did not cause any difference to the training and validation loss nor the convergent time. The CNN-CNN model with the dense layer was not used as well since its output is less ideal than the CNN-CNN model without the dense layer. It always produces a solid tree instead of a stroke-based tree with line boundaries.

User Evaluation

We randomly sampled 52 sketch and text pairs from our testing data and ran them using the following four models: CNN-CNN model without dense layer, CNN-RNN model, RNN-RNN model version 1 2. In total, we generated 208 sketches. 17 users in total were asked to evaluate the 208 generated sketches by answering the following questions: From a score of 1 - 5, how well do you think the transformation did? This question is used to evaluate how well our model transforms the sketch under human perception.

Table 1: **Human Evaluation**. We report on the mean and standard deviation of the ratings of 208 generated sketches given by 17 reviewers.

Model	CNN-CNN	RNN-RNN v1	RNN-RNN v2	CNN-RNN
Mean	2.58	1.73	2.63	1.60
Std	0.96	1.12	1.09	0.93

The second version of the RNN-RNN model and the CNN-CNN model received the highest score, though scores received by all models are not high. All of our models also have relatively large standard deviation, which implies the output quality is not consistent.

Inception Model Evaluation

In order to evaluate our models quantitatively on a larger scale, we trained an Inception Network [12] as a classifier with 8 classes from the Sketch-RNN *Quick, Draw!* dataset. After training with 70 thousand sketch images from each class,

the InceptionV3 model achieves around 92% top 1 accuracy on sketch image classification, so we are confident that this classifier will provide a fair evaluation on whether a generated sketch looks like a tree or not. Because the inception model was trained with images drawn from sketch strokes, we did not use it to evaluate the sketches generated from CNN decoder, which directly generates images on pixel level. According to Table.2, the Inception model evaluation result on 1441 testing images shows that the model with RNN encoder and RNN decoder when fed with original sketch 1 strokes generates the best tree-looking sketches. Technically, this RNN decoder was only trained to reconstruct target sketch strokes by itself, and feeding an existing sketch during inference time does not exactly align with the purpose it has been trained for. However, the fact that RNN decoder can generate 72.8% tree-looking sketches with hints from an existing tree sketch shows its capability of learning the relation between adjacent strokes in a sketch.

Table 2: **Inception Model Evaluation**. We report on the number and percentage of sketches correctly classified as tree by the inception model. All models were evaluated with 1441 testing data.

Model	RNN-RNN v1	RNN-RNN v2	CNN-RNN
number	280	1049	75
percentage	19.4	72.8	5.2

Example Results

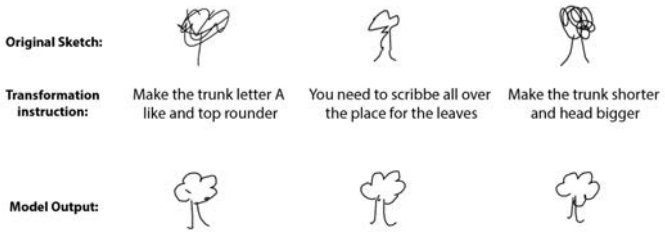


Figure 7: Examples generation of the latent vector transformation model

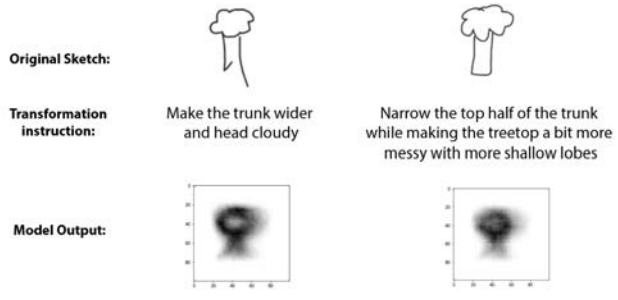


Figure 8: Examples generation of the CNN to CNN model (without dense layer)

Figure 9: Examples generation of the RNN to RNN model (first version decoder)

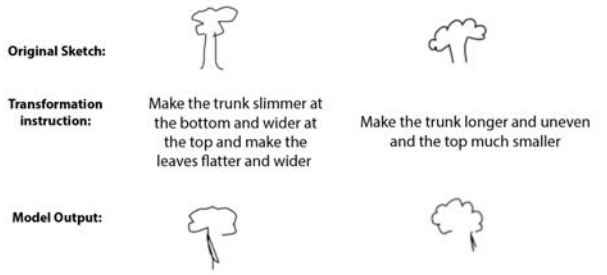


Figure 10: Examples generation of the RNN to RNN model (second version decoder)

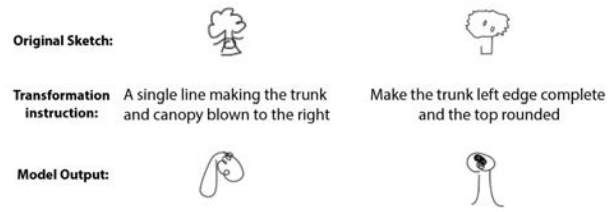


Figure 11: Examples generation of the CNN to RNN model (second version decoder)

CONCLUSION

In this work, we explored different methods to transform one sketch to another sketch given text description that guides the transformation. We used two evaluation methods to assess our model. User evaluation mainly shows whether the generated sketch is tailored to the textual description, and inception model evaluation mainly shows whether the generated sketch looks like an object (in our experiment, the object is a tree). Given both user and inception model evaluation results, we concluded that the model using both RNN as encoder and decoder, and predicts the transformed sketch at a single time step, achieves the best performance. However, the average scores overall are not very high and the standard deviations are a bit large, which implies that there is still much room for improvement. The architectures of our models are relatively simple, and therefore may not capture all features of the sketch and text. A more complex model architecture will probably do better. By using different approaches to solve this unexplored

task, we hope to encourage people to explore further on this new idea, and thus develop better methods to solve it.

FUTURE WORK

1. **Future Applications**: We believe our research will enable many creative applications. By establishing a relationship between natural language and strokes of sketches, many applications in various domains will become possible. In the science and education domain, relations between text and sketches are heavily present. Applying the research idea may help people easily to understand and process the topics in science and education.

2. **Data Improvement**: There are still many problems remaining. Samples are one of the major problems in our research. To have a good natural language model, we need a clean dataset for our model to learn. Since our samples come from human input, there are many noisy cases. Some noisy samples are copied from others without describing the true situations, and some of them are just meaningless sentences. So we need to build a stronger filter to get a cleaner dataset. Also, natural language models need a lot of samples for models to understand the language, and we should get more data if possible to improve the results.

3. **Model Architecture Improvement**: Although we have tried many different types of models, due to the lack of previous researches in this area, our model is not very complex. In the future, models can have much complexity so that it will become more flexible to learn the language. For latent vector transformation, models can continue to improve by adding layers that can extract features of latent vectors which will help models to understand latent vectors easier. For sketch to sketch models, one straightforward method is to stack more LSTM layers in our current model so that the model will contain more parameters to fit the samples. Another way is to adapt LSTM structures or CNN structures from other similar topics which may lead to good performance.

4. **Evaluation Improvement**: The evaluation part is another major problem since our models are generative. Now we are manually evaluating the accuracy of our model. This method is not only inefficient but also contains a lot of bias. One way future work could explore is by using co-teaching ideas[5]. The main idea is to train two networks and select a small set for possibly clean data after fitting all the data each epoch. Researchers may use clean mini-batch data to validate each other and then feed the data forward in the next epochs. This idea may solve part of the evaluation problem and improve the result by making the dataset less noisy.

References

[1] Mathieu Andreux and Stephane Mallat. 2018. Music Generation and Transformation with Moment Matching-Scattering Inverse Networks.. In *ISMIR*. 327–333.

[2] Hong Chen, Ying-Qing Xu, Heung-Yeung Shum, Song-Chun Zhu, and Nan-Ning Zheng. 2001. Example-based facial sketch generation with non-parametric sampling. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, Vol. 2. IEEE, 433–438.

[3] Wengling Chen and James Hays. 2018. Sketchygan: Towards diverse and realistic sketch to image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9416–9425.

[4] David Ha and Douglas Eck. 2017. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477* (2017).

[5] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. Tsang, and M. Sugiyama. 2018. Co-teaching: robust training deep neural networks with extremely noisy labels. In *NeurIPS*.

[6] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[7] Forrest Huang and John F Canny. 2019. Sketchforme: Composing sketched scenes from text descriptions for interactive applications. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 209–220.

[8] Jonas Jongejan, Henry Rowley, Takashi Kawashima, Jongmin Kim, and Nick Fox-Gieg. 2016. The quick, draw!-ai experiment. *Mount View, CA, accessed Feb* 17 (2016), 2018.

[9] Chao Ma, Xiaokang Yang, Chongyang Zhang, Xiang Ruan, Ming-Hsuan Yang, and Omron Coporation. 2013. Sketch Retrieval via Dense Stroke Features.. In *BMVC*.

[10] Elman Mansimov, Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. 2016. Generating Images from Captions with Attention. In *ICLR*.

[11] Alexander J Ratner, Henry Ehrenberg, Zeshan Hussain, Jared Dunnmon, and Christopher Ré. 2017. Learning to compose domain-specific transformations for data augmentation. In *Advances in neural information processing systems*. 3236–3246.

[12] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.