THIS IS A THESIS
OF THE UNDERGRAD VARIETY

Ursula Undergrad

Stanford University
April 2022

An honors thesis submitted to the department of
Civil and Environmental Engineering
in partial fulfillment of the requirements for the undergraduate
honors program

Advisor: Emmy Noether

_____ Date: _____

Emmy Noether   (Thesis Advisor)

Olga Ladyzhenskaya Professor of Engineering

School of Computer Science

_____ Date: _____

Ada Lovelace   (Thesis Advisor)

Professor

Computer Science

Abstract

Acknowledgements

Contents

Chapter 1

Introduction

Robotics have advanced significantly in the past 30 years, and while at the beginning, robots work in factories on assembly line tasks and are far from our daily lives, they have been moving closer to us and into our homes: from iRobot roomba that can sweep floors to Labrador assistive robots that can navigate around and retrieve heavy loads. In the future, we would want to communicate with robots and collaborate with them on tasks, similar to how we smoothly interact with other people in completing daily chores. Out of the many tasks that we can collaborate with robots on, this thesis focuses on drawing. Creative AI, such as using deep learning models to generate paintings and music, has been a popular research domain. Since creative activities are representative of unique human intelligence, numerous works have attempted to replicate the creative process on machines. For example, Pharmako-AI (Allado-McDowell & Okojie, 2020) is a book co-written by K Allado-McDowell and GPT-3 (Brown et al., 2020) through exchanges between the human and the language model. Works like DALL-E, GLIDE, and DALL-E 2 tackle the problem of synthesizing images from text prompts, or short language descriptions, and these generative models have produced many imaginative and inspring art pieces. Since creative activities are carried out not only in solitude, we attempt to investigate at the intersection of human-robot interaction (HRI) and creative AI. Similar to how people communicate ideas to each other, our research is motivated by the goal of producing creative machines that can interact and collaborate with people.

There is a wide range of research directions to pursue in the domain of drawing with robot; on one end, by focusing more on challenges with robot-arm manipulation, one can investigate constructing highly realistic paintings through precise execution of a variety of painting tools. In this work, we focus on generating simple icon-like sketches, similar to the doodles collected through the

Quick,Draw! game, which asks participants to draw an object under 20 seconds[1] (Ha & Eck, 2017). To realize collaboration on drawing, we want to focus on language understanding for robotics and enabling people to participate in the creative process of aforementioned generative models. Doodling avoids learning sophisticated art techniques and allow people without art background to command the robot to draw simple graphics. Our project is interested in learning how people use language creatively to describe the sketches they want to create with a robot. Therefore, one of our main challenges is building a drawing system that can respond to unconstrained language commands, so that, in the future, users of home robots do not have to limit themselves to a set of instructions and can have more natural interactions with the robots.

---

[1]To play the game: `https://quickdraw.withgoogle.com/`

Chapter 2

Related Work

We survey two areas of related work: text-to-image synthesis and sketch representation learning. [change here] There is a long line of works on generating realistic images by learning their underlying distributions, especially on employing generative adversarial networks (GAN). Related to our work is a set of works on generating images from text descriptions or use texts to manipulate image styles, but they often work with a constraint set of language, with the exception of industrial-scale generative models with billions of parameters trained from text-image dataset on the hundred million scale. Most of these [X] not very satisfied with this sentence. work focus on datasets of photo-realistic images, and our work instead centers around sketches, which are abstract since certain features of the objects in the sketches are illustrated figuratively and contain sparse information, as most part of the image is empty. Therefore, we also look into the field of sketch representation learning: what methods have been employed and what datasets are constructed.

2.1 Sketch Dataset

Since our work seeks to collect a dataset of (sketch parts, text description), we survey exisiting sketch datasets and find that they either lack language descriptions, for the entire sketch or for parts in a sketch, or they do not contain semantic object annotations.

TU-Berlin   Eitz et al. (2012) is one of the first works to investigate the characteristics of free-hand sketches and attempt to extract local features based on orietation, which are later used in the task of sketch recognition. It also provides the TU-Berlin sketch dataset that contains $20,000$ sketches spanning 250 object categories with 80 samples in each. The TU-Berlin dataset is then extended for sketch-based 3D object retrieval to contain 1814 new sketches for 130 common household object categories. This additional set also contains hierarchical category labels; for example, the *animal* category contains *arthropod, biped, human, flying creature, quadruped, underwater creature* categories if going one level down the hierarchy. The TU-Berlin contains some of the highest quality sketches for a wide range of categories, but they lack both text descriptions and semantic part annotations, and our dataset contains both types of annotations, although only for simpler sketches in the face and angel category.

Datasets with Text Descriptions   The QMUL dataset contains sketches of shoes and chairs: 419 shoe sketches of various types of shoes, such as high heels, boots, and ballerina flats, and 297 chair sketches of different kinds (Yu et al., 2016). Although QMUL annotates for attributes of the sketches, they come from a fixed set of descriptions and are obtained from product tags, so these descriptions do not reflect how humans would creatively describe the drawings on their mind, which is a feature of our collected dataset. The Sketchy dataset contains $75,000$ sketches of 125 categories, and each sketch is paired with an image; Sketchy is additionally annotated with sketch validity metrics and short descriptions from annotators (Sangkloy et al., 2016). However, these descriptions are more like comments given by the participants, and the collecting process is not carefully designed, unlike our data collection process, where obtaining the text description is the main focus, ensuring the quality of the language. Moreover, the short texts in Sketchy do not describe individual semantic object in the sketches.

Quick,Draw! Dataset   Quick,Draw! collects the largest sketch datasets containing 50 million sketches distributed across 345 object categories, each containing around 100,000 sketch (Ha & Eck, 2017). Annotators for the TU-Berlin dataset have 30 minutes to draw one sketch, and annotators for the QMUL dataset have reference photos for the sketches they would draw later; on the other hand, Quick,Draw! participants had 20 seconds to create the sketches for a randomly assigned category without time to look for reference, so the Quick,Draw! sketches are the simplest; however, since sketches that failed to be recognized by a classification model are filtered, they are in general of good quality and are representative of the general population drawing skills. Moreover, many works on sketch representation learning use the Quick,Draw! dataset, so utilize these sketches allow us to potentially adapt some of the pre-trained generative models. These features of the Quick,Draw! dataset make it very favorable to be used to learn an collaborative drawing robot that can interact

with a wide range of users, including children. However, Quick,Draw! dataset contains neither semantic part annotation nor language descriptions, and our collected dataset contains both in order to build collaborative drawing robots.

Datasets with Semantic Part Annotations   The sketches in our dataset come from the Quick,Draw! dataset, but the original Quick,Draw! dataset does not provide labels for semantic objects in the sketches. The Sketch Perceptual Grouping (SPG) dataset (Li et al., 2018) and the SketchSeg dataset (Qi & Tan, 2019) contain annotations for semantic segmentation, meaning that each stroke in a sketch is paired with a semantic label; for example, strokes in a sketch for the *face* category are assigned one of the following labels: *eyes, nose, mouth, ear, hair, moustache, outline of face*. The SPG dataset annotates for $25,000$ Quick,Draw! sketches, belonging to 25 (out of 345) categories, and it selects 800 out of the original $100,000$ sketches in each category for annotation. The SketchSeg dataset builds upon the SPG dataset by using a recurrent neural work to generate additional sketches stemmed from one sketch in SPG; since each stroke in the generated sketch corresponds to a stroke in the original sketch, it automatically has part annotations. However, since the quality of the generated sketch is dependent upon the generative model, we choose the original SPG dataset, which is also publicly available. The SPG dataset does not contain text descriptions like our dataset, so although they can be used to build a collaborative drawing system, such system would not have the capacity to respond to natural language commands from users, and our work is interested in learning collaborative robots that can communicate freely with people.

2.2   Collaborative Drawing

Sketch-RNN   Along with the Quick,Draw! dataset, Ha & Eck (2017) introduces Sketch-RNN and provides a web demo of collaborative drawing, where users can select a category and draw the first few strokes of a sketch, and the model will complete the rest of the sketch[1]. Sketch-RNN uses a variational autoencoder (VAE) with bi-directional RNN encoder and RNN decoder. Ha & Eck (2017) uses a vector format to represent the sketches: each sketch is a set of strokes, and each stroke is a sequence of points, so the smooth curve is approximated with piecewise linear splines created by connecting adjacent points in the sequence. Each point is a 5-component vector, $(\delta x, \delta y, p_1, p_2, p_3)$: the first two components represent changes in the (x, y) coordinate of the current point compared to the last one; the last three make up a one-hot vector representing current state of the sketch. Let $s = \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix}$, $s = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$, if the current point will be connected with the next point;

---

[1]You can access the demo from this website `https://magenta.tensorflow.org/sketch-rnn-demo`

$s = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$, if the current point is the last point in the current stroke, so the pen is expected to be lifted next; $s = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$, if the current point is the last point in the sketch. The RNN encoder take this sequence as input and uses the final hidden state h of the RNN to parameterize a Gaussian distributions of size N_z, from which a latent random variable $z \in \mathbb{R}^{N_z}$ is sampled. The latent z will be used to (1) initialize the hidden state h_0 of the RNN decoder and (2) be concatenated with the vector at each time step to be fed as input into the decoder. The RNN decoder predicts parameters for distributions (Gaussian mixture model for sampling $\delta x, \delta y$ and categorical distribution for sampling p_1, p_2, p_2) at each time step from the hidden states of the RNN. Since the decoder works in an autoregressive manner, meaning that points in a sketch are passed one by one into the RNN, the decoder can *encode* the first few strokes provided by the users into the hidden vector, which is conditioned on for later steps to produce distributions of points that make up the rest of the sketch. Subsequent works on sketch representation learning can be grouped into those that represent the input in this stroke-point vector format and those that treat the entire sketch as a raster image.

Although our work uses the Quick,Draw! sketches, we render the vectors into raster images in order to use CLIP and its vision-language joint embeddings, since compared to Sketch-RNN, we need to incorporate text descriptions of the sketch parts. It is challenging to align semantic information in text with distributions of the x, y coordinates. While language gives high-level guidance on how strokes are organized on canvas to illustrate a particular concept, the coordinate information is too low level if directly used without an intermediate level of abstraction to aggregate these low-level sequence.

DoodlerGAN   DoodlerGAN (Ge et al., 2020) is a more recent work on collaborative drawing and represents sketches as raster images; it uses Generative Adversarial Network (GAN) to model the sketches [2]. Compared to Sketch-RNN, which can only complete the missing parts of sketches once users are done with their parts, DoodlerGAN generates sketches using a part-based approach so that users and the system can take multiple turns to create sketches more collaboratively. Moreover, as indicated by Ge et al. (2020), compared to vector inputs, raster images can take better advantage of information from spatial structure of the parts and ignore effects of shifting coordinates by focusing more on relationships among the sketch parts. The part-based GAN has two networks: a part selector and a part generator. Given an incomplete sketch with some parts drawn (for example, a bird with its eyes and beak), the part selector determines what kind of part to draw next (for example, the bird wings). Then given the partial sketch and the category of the next part, the part generator produces a raster image of the part and its location on canvas. The part generator is a conditional GAN derived from StyleGAN2, and it takes as input an image with number-of-parts+1

---

[2]Try the DoodlerGAN demo here: `http://doodlergan.cloudcv.org/`

channels: one channel for each part and an additional channel for the whole partial sketch. The part selector uses the encoder of the part generator with an extra linear layer at the end as the classification head.

While the part-based generation aspect of DoodlerGAN aligns with our goal to create collaborative drawing agent, there is no language associated with each part, and the generation process is not guided by language supervision. Therefore, we

SketchBirds

Past works have learned sketch representations from a wide variety of tasks: sketch recognition, sketch generation from image, image generation from sketches, sketch retrieval of 3D objects, sketch retrieval of images, semantic segmentation of sketches, etc. The survey paper by P. Xu et al. (2020) provides a comprehensive overview and systematic taxonomy of . There is a wide range of tasks that can be done on sketches, both unimodal and multimodal, and, for each task, a large reservoir of deep learning methods used to solve the tasks. P. Xu et al. (2020) gives a comprehensive review of the task taxonomy, summarized the unique challenges associated with each individual tasks, and evaluated the different deep learning methods on sketch recognition through a library `TorchSketch` the authors wrote, and it contains implementation for CNN, RNN, GNN, and TCN. The sections that are most relelvant to us are: sketch generation, sketch segmentation. Sketch generation because we are trying to learn a generative model. Sketch segmentation because we are trying to gain insight about how are semantically meaningful units discovered in sketches and what relationships do the parts have with the whole sketch. Similar to images, sketches have hierarchical structure, and we

The hope is that we can leverage previous work on sketch representation learning to gain insights about sketches and how to learn good representations of them. What is unique about sketches compared to regular RGB images from, for example, ImageNet is that (1) sketches are abstract characterisation of the objects, and although humans can recognize and understand a sketch perfectly, they do not necessarily bear big resemblance to their image counterpart; therefore, methods that work well on RGB images, especially generative models like GAN that have successfully generated wide range of images from texts, a realm that we care about, it is not necessarily the case that they can generalize well to our dataset.

In terms of exploring the multimodal sketch generation realm (text-to-sketch synthesis), a recent work is SketchBird (Yuan et al., 2021). This work, similar to ours, deal with the unique challenge of generating sketches from textual descriptions. They setup the task to mimic or as a counterpart to the classic text to image generation on the CUB dataset (Wah et al., 2011). This work is also

representative of a line of work that is based on GAN, unique in the way that it is outputting sketches, closer to the domain that we are interested in. The line of text-to-image synthesis work begins with conditional GAN (Reed et al., 2016), which also reports results on the CUB dataset. But what is slightly in lack for the dataset that SketchBirds collected But to examine the line of GAN work, we can see that AttnGAN (T. Xu et al., 2017) (what SketchBirds is based upon or) One thing we are especially interested in is how these models are able to extract the text features, and how they fuse text features with image features. Moreover what loss is used to encourage the alignment between the image and text domain. In SketchBirds, a bidirectional long short-term memory (Bi-LSTM) network is used as the text encoder. Inspired by AttnGAN, to extract text vectors that are visually aware, SketchBrids trains the text encoder with image-text pairs while minimizing the Deep Attentional Multimodal Similarity Model (DAMSM) loss, proposed in AttnGAN. This loss is calculated based on attention-driven text-image matching score, where matching is between two vectors, one is the vector representing a word in the sentence, and the other is a weighted sum of vectors of image regions, where the weight comes from a matrix of size $T \times 289$ (T being the number words in the sentence, and 289 being the number of image regions), calculated using dot-product similarity between word in the sentence and sub-region in the image. It seems like from quite a few papers, such as G. Xu et al. (2021), fuse the visual and textual space by combining the visual features using weights calculated by dot-similarity between the two modality, or vice versa to achieve cross attention. G. Xu et al. (2021) uses a LSTM+GloVE setup for the unimodal text embeddings.

The SketchCUB dataset collected by SketchBird contains sketches that are more similar to still-life portrait sketches and are very realistic, but sketches in the Quick,Draw! dataset are more similar to icons. This is due to how SketchCUB is transferred from RGB images in the CUB dataset by using open-source holistically-nested network (HED). The SketchCUB dataset contains 200 bird categories with 10,843 images. It includes a training set with 8,326 images in 150 categories and a test set with 2,517 images in the remaining 50 categories.

What are some other ways that we can extract visually informed text embeddings.

StyleGAN-NADA: CLIP Guided Domain Adaptation of Image Generators

Of course, there are other techniques to generate images from texts, namely, leverage large pretrained model such as GPT-3. GPT-3 and DALL-E are particular nowadays for researchers to replicate on their own and try to query the immense feature space for creative art pieces. However, the abstract art style work is not our focus, and while creativity is an interesting future direction, we emphasize the collaborative aspect more than creativity.

In the larger realm of RGB images: Therefore, our dataset will be a good benchmark for how well these models work at capturing the individual semantic components of an object. The reason that we claim this is that some work on GAN's have try to look at how to manipulate certain regions in the images by manipulating the latent space. While this line of work also try to look at how .This area of the work is around facial feature editting. Work such as Semantic Photo Manipulation with a Generative Image Prior (Bau et al., 2019), has an interactive interface where the user can use stroke to indicate where in the image they would want a certain object, and the GAN will generate the objects in that location. "semantic image editing tasks, including synthesizing new objects consistent with background, removing unwanted objects, and changing the appearance of an object". semantic edit on an object. They would apply a semantic vector space operation in the latent space. How our work is different from this work is that: how well the methods can work on sketches and how well can the edits can done through language. [?] Moreover, it seems like we need to have an image already in order to do the manipulation, but for our ideal tasks, we start from a blank canvas.

Chapter 3

Data Collection

3.1 Overview

Imagine the following scenario (inspired by the YouTube channel:[]): Today we are going to draw a smiling ice-cream cone. Okay, we are going to first draw a curve as the top of a big scoop of ice-cream. Next, we will draw a sequence of connected U's to represent the bottom of the overflowing ice-cream. Lastly, we will draw a large upside-down triangle as the cone of our ice-cream.

We want to realize this kind of interactions with a robot, as a companion, so we need to collect a dataset that can help us to get closer to this goal. In order to study this problem, we want to collect human sketches, so the first thing we did was designing an web interface. The leading questions of the data collection process. Our goal is to collect a dataset so that we can learn a model that can interactively draw sketches with users. Therefore, we want to collect the drawing for a single step and a person's description of the drawing. Our design of the interface centers around some key questions:

1. Ensure that the drawing responds to the prompt. The underlying assumption here is that the prompt itself will give us some signals in terms of where the objects in the images might be.

2. From the design side, enforce annotators to breaking the sketch generation process into steps. The worst scenarios is for the annotators to

3. How do we make sure that annotators are breaking the sketches they provide into reasonable

steps? What we mean by reasonable here is the fact that there should be a good correspondence between some parts of the sketch and the language that is used to describe it. Although in our daily interactions, we might say something like "we now draw this" or "we can do this", but from a model learning perspective, or more so as a first step, we want there to be little ambiguity in our language and disallowing words like "this".

Our interface has experienced 2 main versions, and the major difference between the two is that the first one asks users to draw the sketches and annotate each step in their drawings while the second version asks the users to annotate existing sketches. The turning point happens after a pilot deployment of the first version, during which we identified several problems: (1) users take too long to complete one task, and it is outside our budge to collect an ample dataset; (2) users cannot separate the entire sketch into steps consistently, and the annotations either describe more or less than what was done in a single step. In order to shorten the task time and alleviate the burden to think about how to draw certain objects, our second version uses sketches from the Quick,Draw! dataset collected by Google and asks users to provide textual annotations for each part in the sketches. The following sections will walk through each version and discuss the data collected using each version. The following sections will walk through each version and discuss how the design reflects or answers the above N criteria and what in reality happened that caused us to change the design.

Later, we discovered that by simply using existing sketches without asking for users to draw for the prompt would significantly reduce the data collection time, and it would also allow us to put aside DQ 2. In general, if you think about it, classic collection tasks such as assigning label to images/texts or drawing segmentation box, the goal of the task is very clear, and it is easy to determine the quality of the work when you glance at it, or easy to verify. At the beginning, we found it very difficult to describe what should be drawn and what should not be drawn, or what can be written and what cannot be written.

The general trend of the data collection process is that we try to simplify the data collection interface and reduce the number of criteria that we need to satisfy, since each introduces a factor of uncertainty.

Since the beginning of data collection, an important question we try to answer is how do we define a semantic unit in the sketch? The end goal is to achieve the kind of interaction shown in the YouTube video *How To Draw A Cute Ice Cream Cone*, and in it, the instructor oftens uses sentences in the form of "Let's draw a X for Y", where X describes the geometric features of the object Y. For example, "Let's draw *small connected U shapes* for the *bottom of the ice-cream cone*." Therefore, at first, we thought of decomposing the drawing process into a sequence of common geometric shapes,

and the objects that they represent become the basic semantic units. At each step, the annotator is first asked to Version 0 was never deployed. I think at this stage of the data collection, we are trying to decide whether there should be a fixed set of primitive that the users could choose from, so learning the model becomes learning to parameterize, for example, the dimensions of the set of primitives.

Functionality:

- Draw the figure and the page will record the sequence

- User can replay its drawing sequence. The original idea was that users will first create the drawing, and then they can replay the sequence as they annotate for each step.

The very first test version: In terms of the main task, I created a test version to confirm that the idea of the drawing board is sensible.

Press *Record*, Draw on the board, Press *Stop* when done with drawing, *Submit* the drawing if one is satisfied with the quality, *Play* to revisit the drawing, *Cancel* to start over.

What was the original motivation behind this functionality was that it will aid the annotators to review the drawing process and divide it into better steps. Responding to DQ 2. However, in this very crude version, we did not really incorporate features for either Responding to DQ 3,

We begin with a very crude version, and then we decide to add features that can allow us to realize the DQ 2 and 3.

The actual Version 0 has the following flow: There is a practice board, you can try to practice drawing so that the actual drawing submit has good quality and respond to the prompt (reflecting DQ 1). Then hit *Ready to Record*, again baking the sequence into the design of the website will help us to enforce collecting a dataset of steps. Another purpose is to help the annotators decide beforehand what are the necessary primitives used in the process. Why was I so fixated on the primitives, because the abstractness of the icons is what interested me the most. The entire research journey was very explorative, it sorted of started with a sense of *oh, this question or aspect of how humans do things is interesting, I wish robot can do the same.* And what is that thing that I thought was interesting, it was how Rain and I were able to draw the icons and the interactions. The first thing you will do is select a primitive from a list, and then you will draw the step that contains the primitives. Hit *Next* to move on to drawing the next primitive. There are will be a little tag at

the bottom showing what is the primitive that corresponds to the step that is drawn on the board. Repeat until finished and hit *Done*. At the end, again, *Play*, *Submit*, or *Cancel* to start over.

? Should we use primitive shapes for users to choose from? The reason for considering this aspect is whether during generation we want to learn to change parameters of a fix set of shapes or generate un-constrained strokes. For the first option, we want users to compose a drawing with primitive shapes, much like using In order to learn a more general model, we decided that we want to collect strokes instead of fixed primitive shapes, so we moved onto creating a table that accompanies the drawing board, where the user can choose to annotate each step they draw.

In Figure 3.1.

3.2 Prompt-Guided Sketch Text Dataset

3.2.1 Overview

For version 1 onward, we decided to host our website on Amazon Mechanical Turk (AMT), which is a crowd-sourcing website that hosts different machine learning annotation taskss. In the remaining text, we use the word *turker* to refer to annotators that we recruit on AMT; we will also use the word *HIT*, Human Intelligence Task, to refer to a task hosted on AMT. Please refer to AMT FAQs for official definition and answers to questions related to AMT.

We have to design the following sections for the task:

1. an interface containing the main task

2. instructions and requirements to describe the tasks and specify what the annotators should and should not include in the annotations

3. a qualification task accompanying the main task to train the turkers to produce high-quality annotations

Compared to Version 0, which we only dabbled with 1 in the above list, we went through all three stages for Version 1 and eventually deployed a pilot. After deployment, we realized that a few major problems from this design: (1) due to the subjective nature of drawing, it was hard to understand in
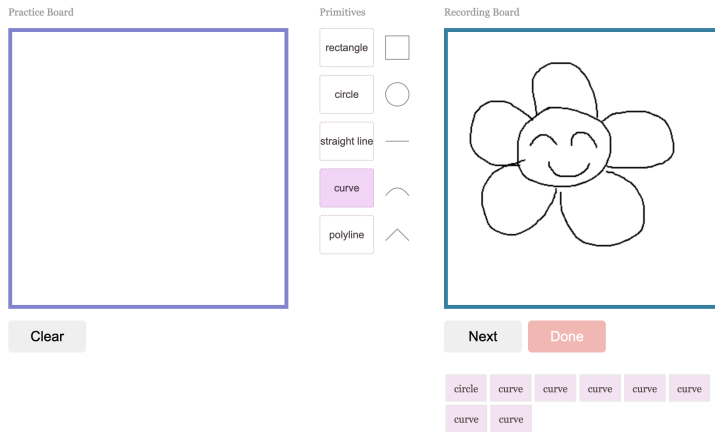
Annotation Instructions

Select the primitive used in step No.1.

Practice Board   Primitives   Recording Board

rectangle   ☐   Done

circle   ◯

straight line ──

curve   ⌒

polyline ⌃

Clear
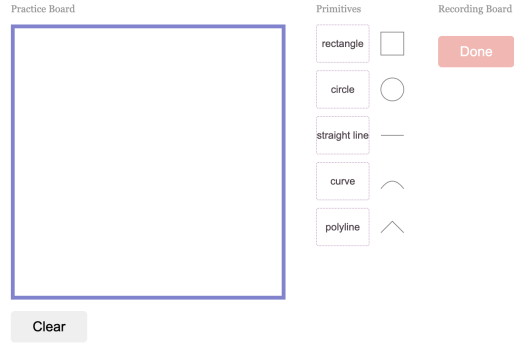
(a) Design of main task for third pilot.

Draw the component using the primitive selected step No.9. Press `Next` when you are ready to move on to the next step.

Practice Board   Primitives   Recording Board

rectangle   ☐

circle   ◯

straight line ──

curve   ⌒

polyline ⌃

Clear   Next   Done

circle   curve   curve   curve   curve   curve

curve   curve

(b) Design of main task for final task.

Figure 3.1: Progress of the design two for the main task in version two.

what ways some annotators are illustrating the given prompts, thus making it difficult to determine the quality of the annotations; (2) turkers are taking more than 30 minutes for each task, showing that providing both sketches and descriptions are inefficient; (3) some turkers are unable to provide descriptions that align with the objects they meant to annotate for; for example, in one step, they drew both eyes and hair, but they only annotate "big eyes".

3.2.2 Interface Design

Main Task

Compare to version 0, we make the following changes to the task interface:

1. Since turkers are paid based on time spent on the task, we decided to forsake the functionality related to the recording and replaying the drawing board.

2. Since we decide to not limit the drawings to be compositions of basic geometric objects, we removed the step to select primitive shape preceding drawing each component.
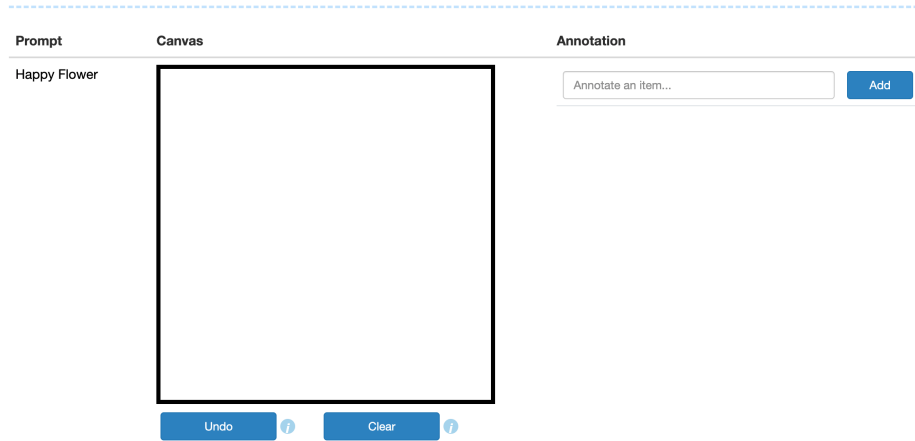
(a) Main task interface at the start of annotation.
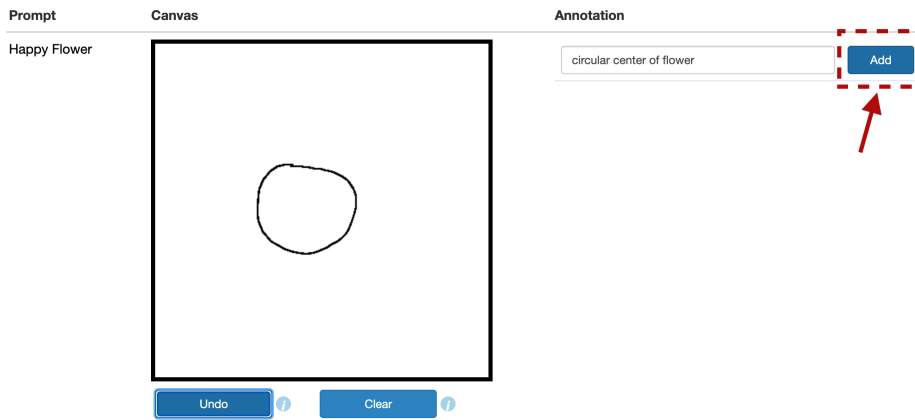
(b) Main task interface before adding text descriptions for the drawing in the given step. Red arrow and box show where to click to add the text.
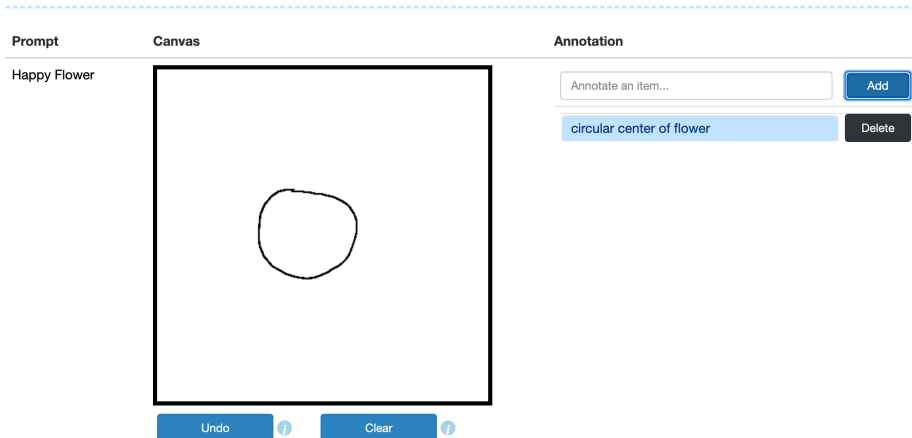
(c) Main task interface after adding text descriptions for the drawing in the given step.

Figure 3.2: The "drawing-and-adding" functionality in Version 1. Repeat the above process for each step in a sketch.

We illustrate a typical annotation process with Version 1's interface in Figure 3.2. The annotator starts with an empty canvas and empty table for textual descriptions, as shown in Figure 3.2a. For the annotator's convenience, we include a *Undo* button and a *Clear* button for erasing strokes and clearing the entire canvas. Then, the annotator draws a step in the sketch, and they would need to enter the text description for this step into the *Annotation* column and hit *Add* to display it as a new row in the annotation table. (Figure 3.2b and 3.2c show what the annotation table looks like before and after adding the texts for a given step.) If the annotator wants to remove an entire step, the objects in the drawing and the text description, they can use the *Delete* button next to the texts to do so. An example is shown in Figure 3.3. Repeat the drawing-and-adding process until the drawing is done. The design of a table for adding and deleting text annotations intends to encourage turkers to break their drawing into a series of semantically meaningful parts, responding to principal 3 and 1. Enabling users to be able to delete each component is also demonstrative of these two principals.

We encountered some difficulties when implementing the *Delete* functionality. At the beginning, we treated erasing strokes as drawing the same strokes but in white color; however, when strokes overlap each other, overwriting with white strokes would break other strokes into segments. There-fore, we designed the drawing canvas to use layers like Photoshop, so that deleting strokes would be the same as deleting an entire layer, thus leaving other strokes intact.
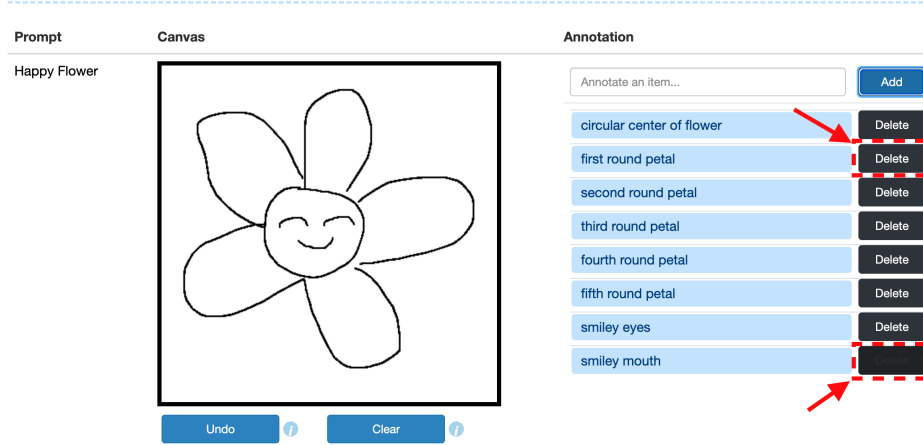
(a) A complete annotation for the prompt *Happy Flower*. Red arrows and boxes point to *Delete* buttons that can be used to delete the steps associated with the textual annotaitons, if the annotator is not satisfied with the steps.
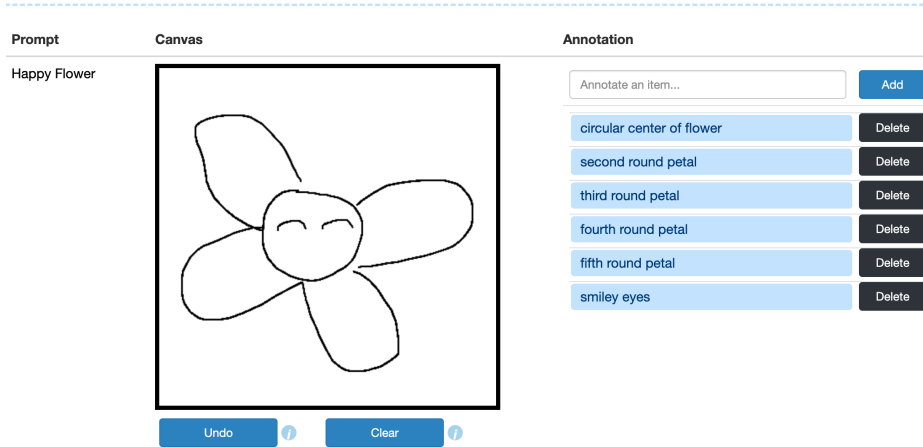
(b) Main task interface after the two steps associated with *first round petal* and *smiley mouth*, respectively.

Figure 3.3: Demonstrating the functionality of the *Delete* button.

Instruction and Requirement

We show the layout of the instructions in Figure 3.4. We begin the instruction by giving a short explantion on the motivation behind collecting this dataset to prime turkers for good-quality annotations, since they would understand more about what the purpose for collecting this dataset. What we struggled the most when drafting the requirements was deciding what would be a single *step* in drawing and how do we clearly explain this definition to the turkers? In Version 0, we relied on common geometric shapes to decompose a drawing into a sequence of steps. In Version 1, we considered asking turkers to annotate for each stroke in the drawing, but we quickly ruled out this

Instructions   +-

Most of the current image synthesis methods output an entire image at once, but we are interested in creating an agent that can doodle like humans: drawing one
component/object at a time. For example, when people are asked to draw "a house", they are most likely to first draw the roof, then the body of the house, the windows, the
chimney, and lastly the door. We would like to collect a dataset of doodles done by people in response to different prompts; moreover, we would also like to collect people's
annotations for every object in the doodle.

In the actual HIT:

- You will see the phrase describing what to draw in the **Prompt** column.
- You will create a drawing that illustrates the prompt in the **Canvas** column.
  - A **stroke** is created between the time you push down, drag, and lift up your mouse.
  - You can draw multiple strokes in one step, but only one object in one step. (More details can be found in the Requirements section below.)
- During drawing, you should annotate each object in your drawing in the **Annotation** column.
  - Add the annotation with the Add button.

Figure 3.4: Instruction section of Version 1.

option since it was time-consuming, and it did not align with how the instructor taught the child in the *How To Draw a Cute Ice-Cream Cone* video. We decided that turkers should annotate for each *object* in the drawing. The ambiguity around the word *object* has posed the biggest challenge in defining a clear set of requirements. For example, when drawing for the prompt *Happy Face*, one reasonable decomposition is annotating for 4 steps: face, eyes, mouth, and the face contour. However, for someone who draws very detailed eyes, they might want to annotate for the shape of the eye socket and the length of the eye lashes. So what level of specificity should be allowed? There is a wide spectrum of allowed annotations depending on how a person approaches drawing for the give prompt. Indeed, the great variation and uncertainty that comes from individuality and personal styles demonstrated through drawings would eventually drive us to not collect drawings and simply ask for text annotations for sketches found in existing datasets. At the time, we resorted to repeatedly testing the interface with lab mates to refine the requirements. Here is an excerpt from an old version of the instructions, in which we tried to explain a single *step* of the annotation:

In each task, we show **1 prompt** from which we would like to get

1. A drawing containing 1 **entity** that you think illustrates the prompt.

2. Text annotation for every **"component"** that makes up the entity. The word "component" is intentially vague, and it depends on how you compose your drawing. For example, in the above example, the prompt is "smiley face", and during the process of creating a "smiley face" entity, we used 4 components: a face, a left eye, a right eye, and a mouth. For each component, you can annotate with "face", "left eye", "right eye", "mouth", respectively; you can also annotate with more details describing the shapes of each component: "face that looks like an arc opening downwards", "a left eye that is an arc", "a right eye that looks exactly like the left eye", "an arc-like mouth". Try to use creative and descriptive languages. You can draw a component using multiple strokes.

We also need to come up with examples explaining each requirement. We select a few major subversions of the requirements resulted from circulating the interface within our lab.

Requirements and selected examples used in the first release in lab (Item 1 to 3 meant to enforce principal 2; Item 4 to 6 for 3 and 1):

1. Do not draw entity that does not respond to the prompt. For example, given the prompt *Smiley Face*, the drawing should not contain irrelevant objects like a house.

2. Do not draw more than one entity that responds to the prompt. For example, One should not draw two *Smiley Face* entities, although each *Smiley Face* entity is good. However, you can draw multiple tree objects to illustarte the prompt *Forest*.

3. Do not draw entity that is ambiguous in terms of illustrating the prompt. For example, the drawing (Figure 3.5a) looks more like a *Sad Face* than *Smiley Face*.

4. Do not draw one component that contains more information/content than what the annotation for that component describes. (A counterexample is illustrated in Figure 3.5b.)

5. Do not split the drawing of a component into multiple steps, unless you can annotate each step separately. (A counterexample is illustrated in Figure 3.5c.)

6. Do not annotate a component more than once.

(a) An example included in the first version of the requirements, explained in more details in item 3.

Requirements and selected examples used in the second release in lab (Item 4 is meant to enforce principal 2; the other items for 3 and 1):

1. Draw *one* item at a time and provide its corresponding annotation. For example, the text annotation says "left eye", but two items are drawn: a left eye and a right eye.

2. The annotation should describe its corresponding item in the drawing *entirely*.

3. The annotation should name the item.
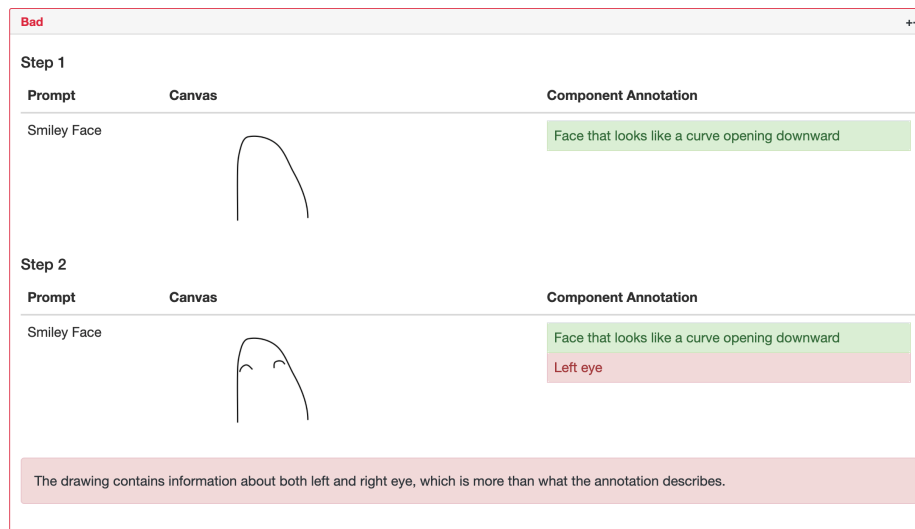
(b) Unaligned drawing and text description.
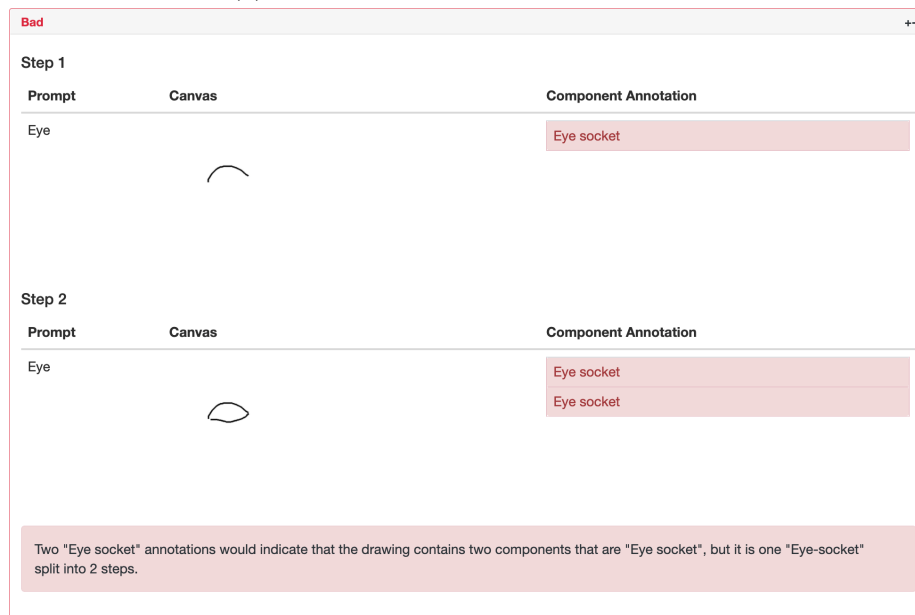
(c) An example of misalignment: text description *overflow* into multiple steps.

Figure 3.5: Screenshots of counterexamples used in first version of the requirements in Version 1.

4. Desired properties of good drawings:

   - Contain as many items as possible, but be sure that they all contribute to illustrating the prompt. For example, draw more than just two eyes for a face.

- Use shapes creatively. For example, draw a triangle for the left eye, and annotate accordingly with "triangular left eye that shows suspicion".

5. Desired properties of good annotations:

  - Use descriptive languages. For example, "a left eye that looks an arc facing downward".
  - Include the intention/purpose of drawing an item. Explain in the annotation reasons for drawing the item. For example, "thumbs-up next to the face that really shows how happy the face is".

Requirements and selected examples used in the third release in lab (Item 1 is meant to enforce principal 2; the other items for 3 and 1):

1. Each drawing should contain at least 2 steps.

2. Annotation of each step should include at least the name of the drawn object(s).

3. If draw multiple copies of the same object, draw each object in a separate step and give different annotations by using, for example, cardinal or ordinal numbers. (An example shown in Figure 3.6)

4. Differentiate between plural and singular forms.

5. The name of the whole should not be used for its parts. (An example shown in Figure 3.6)

6. The word "right" always refers to this side: $\implies$
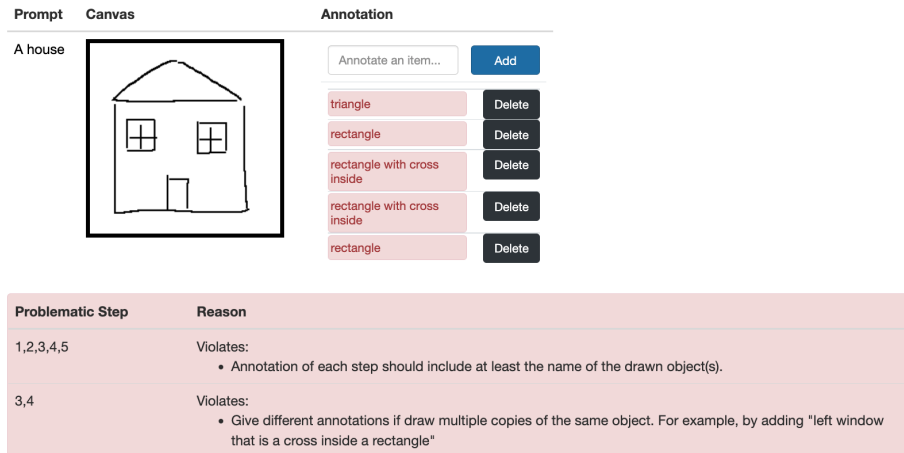
Figure 3.6: Screenshots of counterexamples used in third version of the requirements in Version 1.

After a series of smaller changes, we eventually arived at the final version of the requirements for Version 1, as shown in Figure 3.7. Most of the requirements are dedicated to ensure principle 3 and 1. Requirement 1 ensures that no irrelevant sketches and trivial annotations are provided, and we resort to good faith that the annotators would provide a sketch that illustrates the given prompt. As expected, problems related to ambiguous sketches and unaligned text annotations surfaced after the deployment on AMT, eventually resulting in a complete change in format and lead to Version 2.
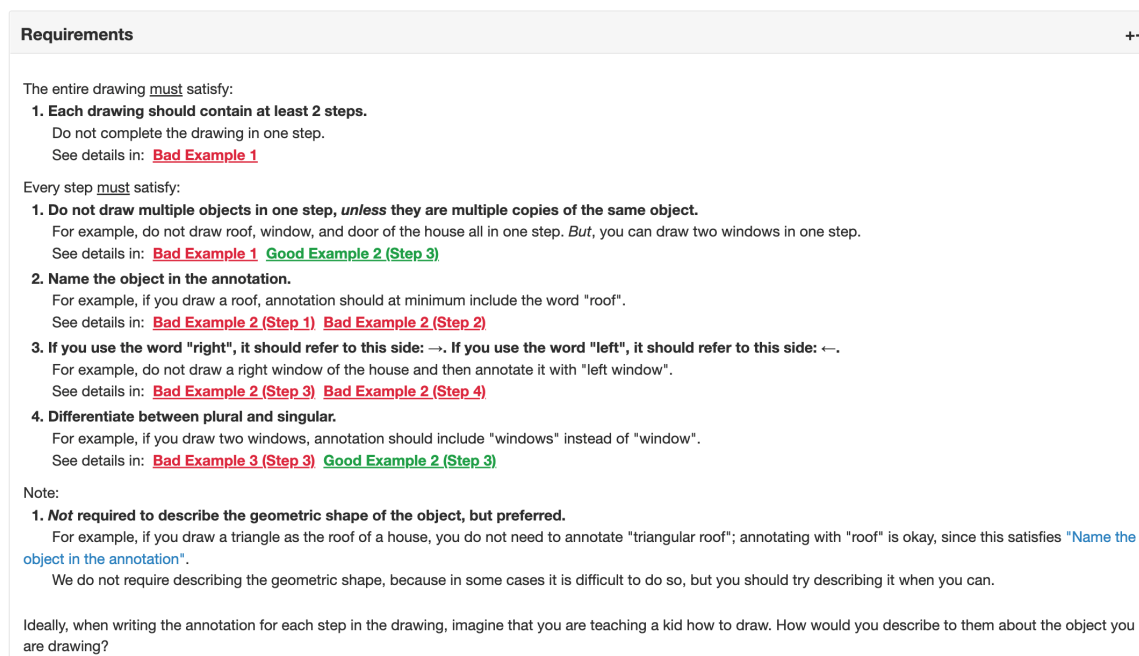
Figure 3.7: Screenshots of final version of the requirements in Version 1. The Bad Example links to counter-examples of the requirements, and Good Example links to good examples. When turkers click on the links, they are directed to the examples illustrating the corresponding requirement.

We also show Bad Example 2 in Figure 3.8 as an instance of the examples used in the final requirements. To view all the examples, refer to: `https://erinzhang1998.github.io/portfolio/amazon_anno`.

Qualification

We setup a qualification test on AMT to (1) train turkers to have better understanding of the task and (2) to select turkers who can provide annotations that satisfy all the requirements. Similar to the process of writing the requirements, we went through several rounds of testing with students in the lab to come up with a set of questions that have good correspondence with the

requirements. The qualification test starts with the same instruction and requirements that will be used in the final HIT, thus allowing turkers to familiarize themselves with the requirements; moreover, this give them a chance to ask for clarifications before the final HIT. The test leads with a navigation bar (Figure 3.9) to make it convenient for turkers to switch between questions; originally, we displayed all questions in one page, but some people found it time-consuming to scroll from the later questions back up to the instructions, so we decided to display one question at a time. We show one question from the final qualification in Figure 3.10. We replicate the exact main task interface in the qualification test, and turkers need to determine whether every step of the mock annotation satisfies all the requirements; we also include hints on which requirement the question is testing for to encourage turkers to revisit the requirements and form better understanding of the task. To see the full test, refer to: `https://erinzhang1998.github.io/portfolio/amazon_qual`.

3.2.3 Deployment Results

To deploy our first pilot, we need to come up with a set of prompts that are in the forms of *adjective×noun*. The list of adjectives includes: *happy, sad, surprised, sleepy, lovestruck, evil*; the list of nouns includes: *person, kid, cat, bear, dog, sheep, jellyfish, cup of boba, apple, burger, sun, moon, star*. We hope to test what drawings and text descriptions annotators would provide for prompts that ask for imaginative beings not in this world, such as *evil apple* or *lovestruck moon*. Our first reason for doing so was that current text-to-image synthesis models, such as DALL-E and GPT-3, can produce creative artwork from abstract prompts that include novel compositions of unrelated concepts; we want to create a dataset that has the capacity to support learning models that can similarly respond to these imaginative prompts through interactive drawing. [!] Moreover, if we backtrack to version 0 for a second, the reason why we considered basic geometric shapes was because we are interested in how humans are able to transfer the usage of a circle to different context: a large circle could be a face, an eye, a big piece of cherry, or a moon, so transferring the same visual concept to different sketches. Also there is an aspect of transferring the same language to different context, such as in what ways the adjectives demonstrate the same concept across different object and in what ways they adapt and show different visual qualities when used on different objects.

[Figure v1.results.4: drawings from the amt pilot]

What surprised us was the amount of time turkers spent on the task. Histograms of time each annotator spent on the task is illustrated in Figure v1.results.1. Statistics of the distributions are shown in Table v1.results.1. The discrepancy might be caused by the fact that lab members with their background in computer science have an implicit understandings of what kind of quality data

are needed to train ML models.

[Figure v1.results.1: a: oct 28 lab deployment. b: dec 28 amt deployment] [Table v1.results.1: comparing the statistics of lab vs. amt deployment]

In violation of DQ 2. Drawing does not illustrate the prompt well. The quality of the drawings are greatly influenced by how well the annotator can understand the prompts. Drawing is by its nature very subjective, so when we were examining through the sketches that we collected, we were not able to understand in what ways some sketches convey the prompts. [Figure v1.results.4: some examples of sketches that cannot illustrate the prompt from our perspective]

In violation of DQ 3 and 1. Another problem was that annotators often fail to describe every parts they drew in one step, or the descriptions miss some parts in the step, or the description does not align well with the drawings. [Figure v1.results.5: some examples of mis-aligned descriptions]

3.3 Contrastive Sketch Text Dataset

3.3.1 Overview

In response to the pilot results, we reconsider the data collection pipeline to reduce the uncertainty around collecting drawings that illustrate the prompts and textual descriptions well-aligned with each step in the drawing. Firstly, in order to alleviate the burden of drawing from the annotators, we examined existing sketch datasets, and information regarding the advantages and disadvantages are shown in Table v2.datasets.1. [Table v2.datasets.1: pros and cons, stats of different sketch datasets]

Between Sketch Perceptual Grouping (SPG) and SketchSeg, both containing annotaiton for semantically meaningful parts in sketches, SPG annotates for QuickDraw sketches while SketchSeg collects its own sketches. We picked SPG, since it will be easier in the future to extend our datasets given the large QuickDraw reservoir of sketches. Moreover, SketchSeg dataset contains a *fourleg* category that includes many different kinds of animals, such as horse, sheep, and cow, but the QuickDraw categories are more fine-grained, so form a model learning perspective, SPG will also be more generalizable. Therefore, to combine our previous goal, collecting sketches that illustrate certain *adjective×noun* prompts, we decided to provide annotators with the sketches from QuickDraw and ask them to annotate for each semantically meaningful part provided in the SPG Dataset.

In order to avoid dealing with discrepancies between performance of fellow graduate students and that of the turkers, we deployed a short pilot test of the new version and identified the suitable format and areas that need written requirement for the turkers to avoid these mistakes. Therefore, the transformation of the task format is driven by mistakes we have seen during the pilot trials.

Main Task

In Figure 3.11, we show the transformation from our first pilot of version to the final task that is deployed to collect the entire dataset. Overall, we used non-gray colors to highlight the parts in the sketch that we want annotations, another design to help with annotation speed. For simplicity, we restricts the annotations from whole sentences (Figure 3.11a) to only adjective phrases (Figure 3.11b, 3.11c, 3.11d). The benefit of juxtaposing two sketches and simultaneous annotate for two sketches is that annotators are implicitly encouraged to provide descriptions that identify features of the objects that differentiate the two sketches. This method is also proposed to facilitate the annotation process and to take less time, since it is easier to differentiate and perform a contrasting task than to generate descriptions from a single sketch. At the beginning, we explicitly mention that the goal of the task is to describe the differences between the objects in the sketches (*Describe differences* in 3.11a and *Compared to Sketch 1/2* in 3.11b), and we received many annotations that contain comparative and superlatives, so we eventually only have a blank without any introductory phrases to overly emphasize that the goal of the tasks is to create a dataset of contrastive pairs of descriptions, and the juxtaposition is meant only as a mental hint to ease annotation.

Instructions

At first the set of instructions was very restrictive and limit the annotators to pay attention to three types of differences: shapes, size relative to other objects in the same sketch, position relative to other objects in the same sketch. The general trend of the changes to the instructions is that we only require that the annotators fill in the blank with adjective phrases and try to not put too much restrictions on the language, in order to achieve our goal of building a dataset with free-form language instructions.

In this version, the advantage is that since we have greatly simplify the task to only providing the textual descriptions, the turkers do not have to spend time coming up with drawings for a $adjective \times noun$ prompt, and they do not have to put effort into keeping track of their drawing process to decide how to divide the drawing process into steps and then annotate for each step.

Essentially, they only have to do the last step. Therefore, the requirements are much easier to write, and we do not have to specify anything in terms of providing drawings that correspond well with the prompts and providing annotations that align well with the drawings in the each step. One thing we tried was to somewhat rely on the examples to give an idea of what kind of annotations we want. Some examples that we used in the tasks are shown in Figure . However, the downside for doing so is that the vocabularies used in by the annotators are primed by those in the examples, and we see that annotators would tend to repeat these vocabularies. Therefore, we especially added the requirement that states the annotators are not limited to words used in the examples, and they should use any words that can illustrate the parts well. The full set of requirements used in the final version is shown in Figure 3.13.

The requirement that was a bit challenging for people to understand was the one regarding *Do not use adjectives related to personal opinions, such as random, good, messy, beautiful, and strage, that are hard to achieve consensus if others were to validate your answers.*. Since we hope that the model can get signal from the texts about what kind of figures to draw, words that do not directly convey visual properties of the parts are not helpful. We later changed the wording to *Do not use adjectives that fail to describe specific visual properties of the objects in the sketches.* A slight caveat here is that we actually hope to collect descriptions that describe the emotions expressed in the sketches. We know beforehand that we hope to collect a dataset for the *face* category, so it is quite common for faces to express emotions like happy and sad, and we were slightly worried that some turkers might consider these words as not illustrating enough visual properties about the drawings, since they are quite abstract, at least compared to adjectives like *rectangular* or *wide*.

Qualifications

We prepared 10 qualification questions, all in the style of yes/no questions. We will use the qualification test to filter annotators who have read through all the instructions and examples and have formed a good understanding of the task. The 10 questions are shown in Figure 3.14. We provides hints in each questions that explicitly state which requirement and examples are helpful for solving the questions. The purpose of the qualification test is not to trick annotators but to ensure both quality and speed of the annotations.

We released n copies of qualifications, and n_2 annotators scored 90 or higher. The average score for the entire test is x, and the rate of correct answer for each question is shown in Table 3.1. Before releasing the qualification, we have tested the test on

Question Number	1	2	2	2	2	2	2	2	2	2
Correct Rate	1	2	2	2	2	2	2	2	2	2

Table 3.1: Success rate of each question in the qualification test

3.3.2 Results

Pilot 1

In order to work out the data collection process, we chose the angel category and try to manually examine the sketches and categorize them based on

One purpose of the pilot is to estimate the amount of money that we need to spend for each task, and from Table 3.2, we see that []

	Max.	Min.	Mean	Med.	Std.
Feb 01 Pilot	1	2	2	2	2
Feb 04 Pilot	1	2	2	2	2
Feb 08 Pilot	1	2	2	2	2
Official Collection	1	2	2	2	2

Table 3.2: Comparing time statistics of pilot task

For the data collection process, we decide to collect for the face category of the QuickDraw dataset, and the reason for it was mainly to echo the choice of many SOTA generative modeling works that are done on the CelebA dataset. It seems that face generation is quite a starting point for many of the generative modeling work. We have also surveyed some text-to-image synthesis methods that use datasets like (1) CUB dataset (2) MNIST (3) Omniglot. Several sketch datasets include the one from DoodlerGAN and SketchBirds. A lot of the datasets focus on one or two categories, so we decide to do the same to ensure that with our budge, we can collect a dataset that contains enough signal to train a generative ML model.

Clustering the faces, we strive to present to the annotators pairs of faces that are distinct as possible in order help them to provide good annotations. It is easier for them to grasp and understand the features of the objects if two sketches are presented in a contrasting way.

If we use CLIP to extract the visual features for the entire face sketch.

The heuristic that we use to choose how to pair up

3.4   Dataset Summary

Our dataset comprises of Quick,Draw! sketches and language descriptions of each semantically meaningful part in the sketch. The dataset contains 2 categories: face and angel, and these categories correspond directly to those in the original Quick!Draw! dataset. The part annotation comes from the SPG dataset (Li et al., 2018). For the angel category, we annotate for the parts *halo*, *eyes*, *nose*, *mouth*, *body*, *outline of face*, and *wings*. For the face category, we annotate for the parts *eyes*, *nose*, *mouth*, *hair*, *outline of face*.

	Face	Angel
Number of constrastive pairs	2515	3060
Number of distinct words	833	1107
Number of sketches	572	787

Table 3.3: Statistics of the dataset by category.
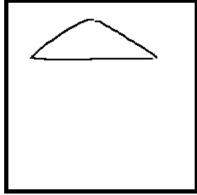
	Face					Angel						
	eyes	nose	mouth	hair	face	halo	eyes	nose	mouth	face	body	wings
Number of sketches	334	572	572	104	572	558	114	8	80	732	781	779
Number of distinct words	228	360	325	152	314	365	112	21	88	379	425	534
Number of con-strastive pairs	689	401	687	126	612	559	114	8	80	733	785	781

Table 3.4: Statistics of the dataset by sketch parts.

In Table 3.4, we see some statistics about the dataset broken down by sketch parts, while in Table 3.3, we all list out the same statistics for the entire face and angel category. In general, we observe that compared to previous work that tend to have a fixed list of adjectives for each object parts, the descriptions in our dataset are free-form and non-constrainted. This characteristics is desirable and aligns with our goal to allow robot to collaborate smoothly with humans, since different people would describe the same things in very diverse ways.

Figure 3.8: Screenshots of an example in final version of the requirements in Version 1.

Qualification Test

Each question is an example of how someone, given a prompt, would draw and then annotate for the object at each step.
You will need to determine if a step satisfies all the *must* satisfied requirements by clickling "Yes" or "No".

Note that:
  - Strokes drawn in the current step are shown in black while strokes drawn in previous steps are shown in gray
  - Annotation for the current step has border around the box
For a question to be count as correctly answered, you will need to correctly determine the validity of **all** steps.

| Question 1 | Question 2 | Question 3 | Question 4 | Question 5 | Question 6 | Question 7 | Question 8 | Question 9 | Question 10 |

Figure 3.9: Screenshots of the navigation bar in the qualification test of Version 1.

Step 1

Q: Does this step satisfy all the requirements?   ☐ Yes   ☐ No
Hint: this step is intended to test your understanding of Requirement 1. Check Bad Example 1

Prompt	Canvas	Annotation
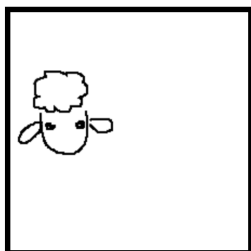
sheep

Annotate an item... | Add

sheep head | Delete

Step 2

Q: Does this step satisfy all the requirements?   ☐ Yes   ☐ No

Prompt	Canvas	Annotation
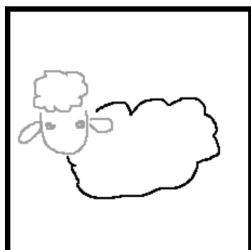
sheep

Annotate an item... | Add
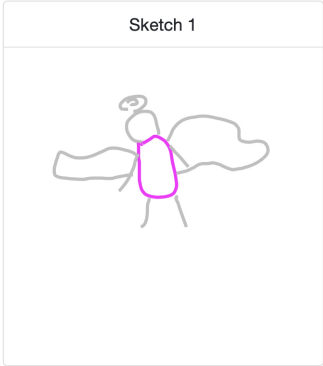
sheep head | Delete
sheep body that looks like fluffy cloud | Delete
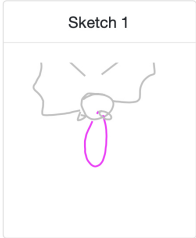
Figure 3.10: Screenshots of question 9 in the qualification test of Version 1.

Annotation 1

Sketch Category	Sketches
angel	

Describe differences between the **angel bodies** (strokes in **magenta** color) in the two sketches.

(a) Design of main task for first pilot.

Sketch Category	Sketches
angel	

Q1: Compared to Sketch 2, Sketch **1** draws a/an ⬚ **angel body** (strokes drawn in magenta color).

Q2: Compared to Sketch 1, Sketch **2** draws a/an ⬚ **angel body** (strokes drawn in magenta color).

☐ If someone is shown the two sketches, the person can pick out one sketch based on the provided differences.
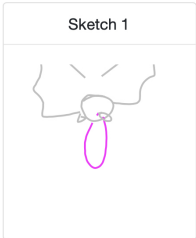
(b) Design of main task for second pilot.
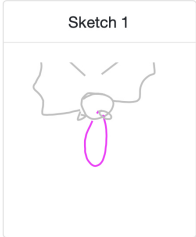
Sketch Category   **Sketches**

angel

Q1: Compared to Sketch 2, Sketch **1** draws a/an

angel body (strokes drawn in magenta color).

Q2: Compared to Sketch 1, Sketch **2** draws a/an

angel body (strokes drawn in magenta color).

☐ If someone is shown the two sketches, the person can pick out one sketch based on the provided differences.

(c) Design of main task for third pilot.

Sketch Category   **Sketches**

angel

Q1: Compared to Sketch 2, Sketch **1** draws a/an

angel body (strokes drawn in magenta color).

Q2: Compared to Sketch 1, Sketch **2** draws a/an

angel body (strokes drawn in magenta color).

☐ If someone is shown the two sketches, the person can pick out one sketch based on the provided differences.
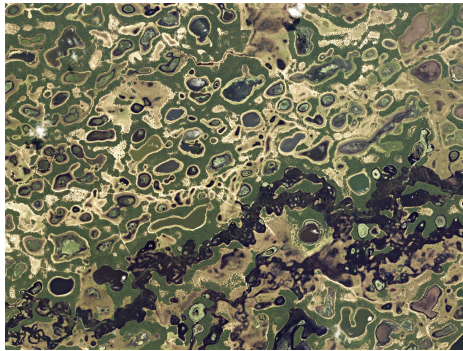
(d) Design of main task for final task.

Figure 3.11: Progress of the design two for the main task in version two.
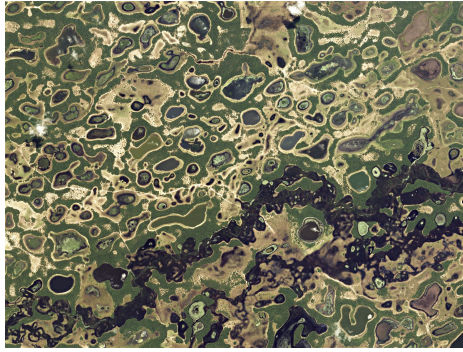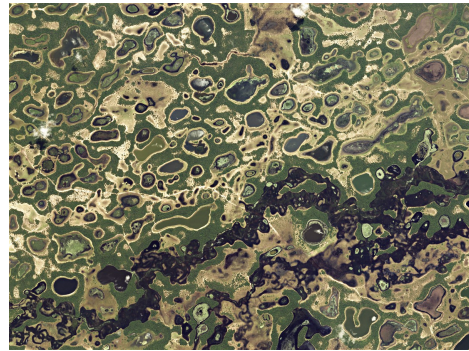
(a) Design of main task for first pilot.
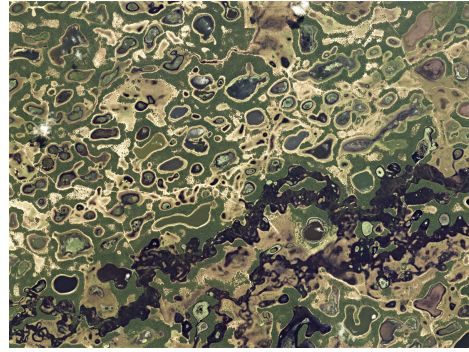
(b) Design of main task for first pilot.

(c) Design of main task for second pilot.
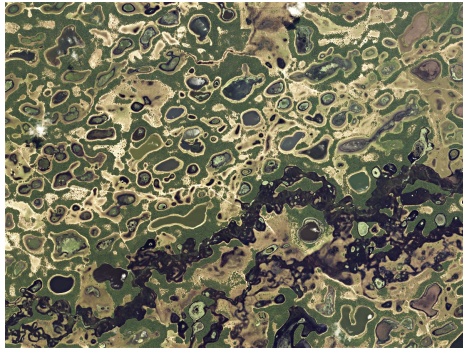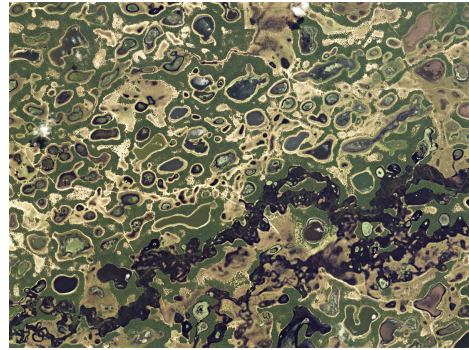

(d) Design of main task for second pilot.

(e) Design of main task for second pilot.

(f) Design of main task for second pilot.

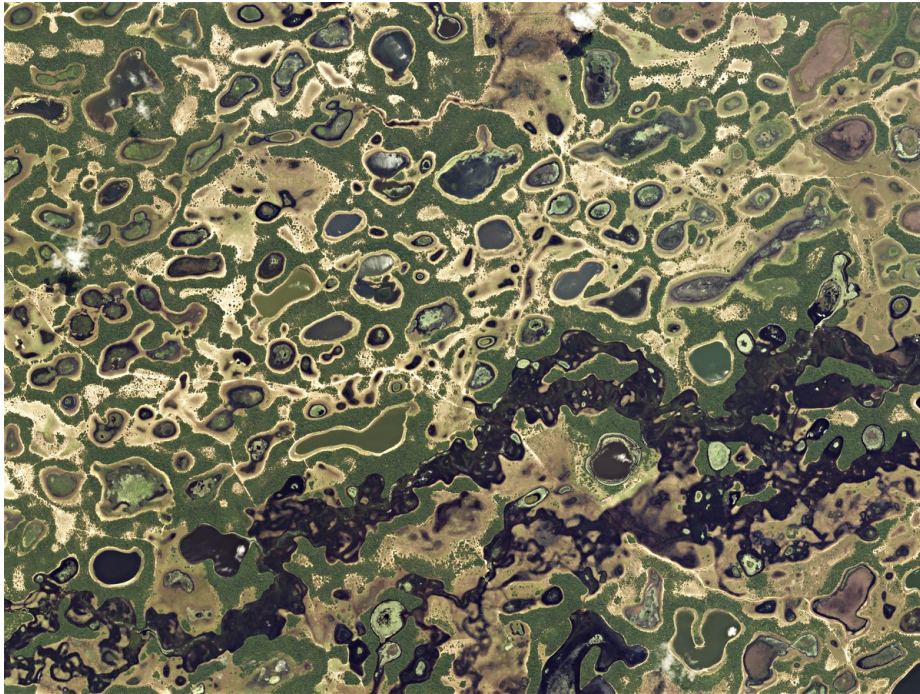Figure 3.12: Progress of the design two for the main task in version two.

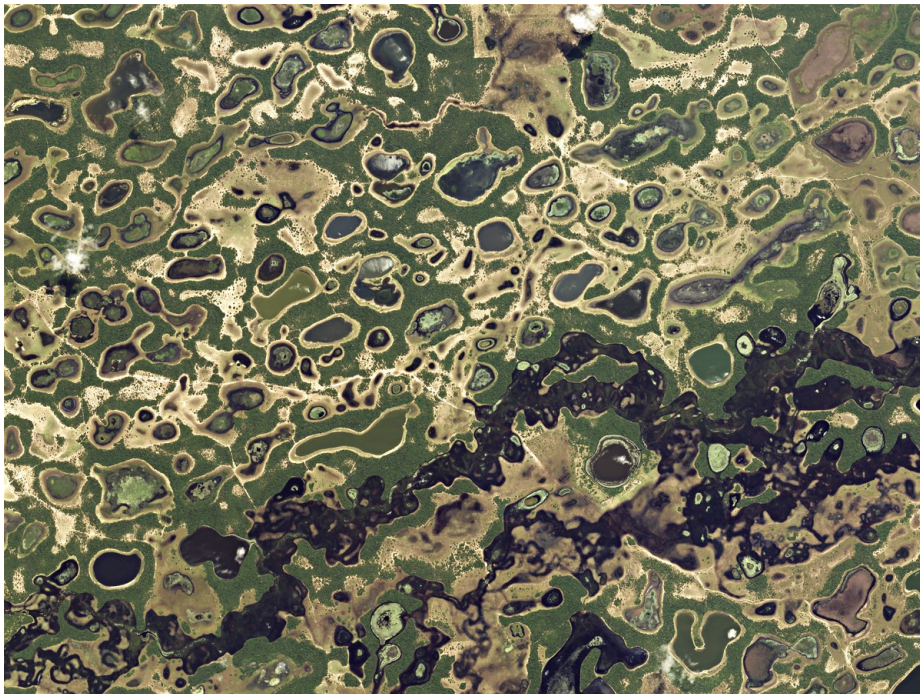Figure 3.13: The set of requirements used in the final task.

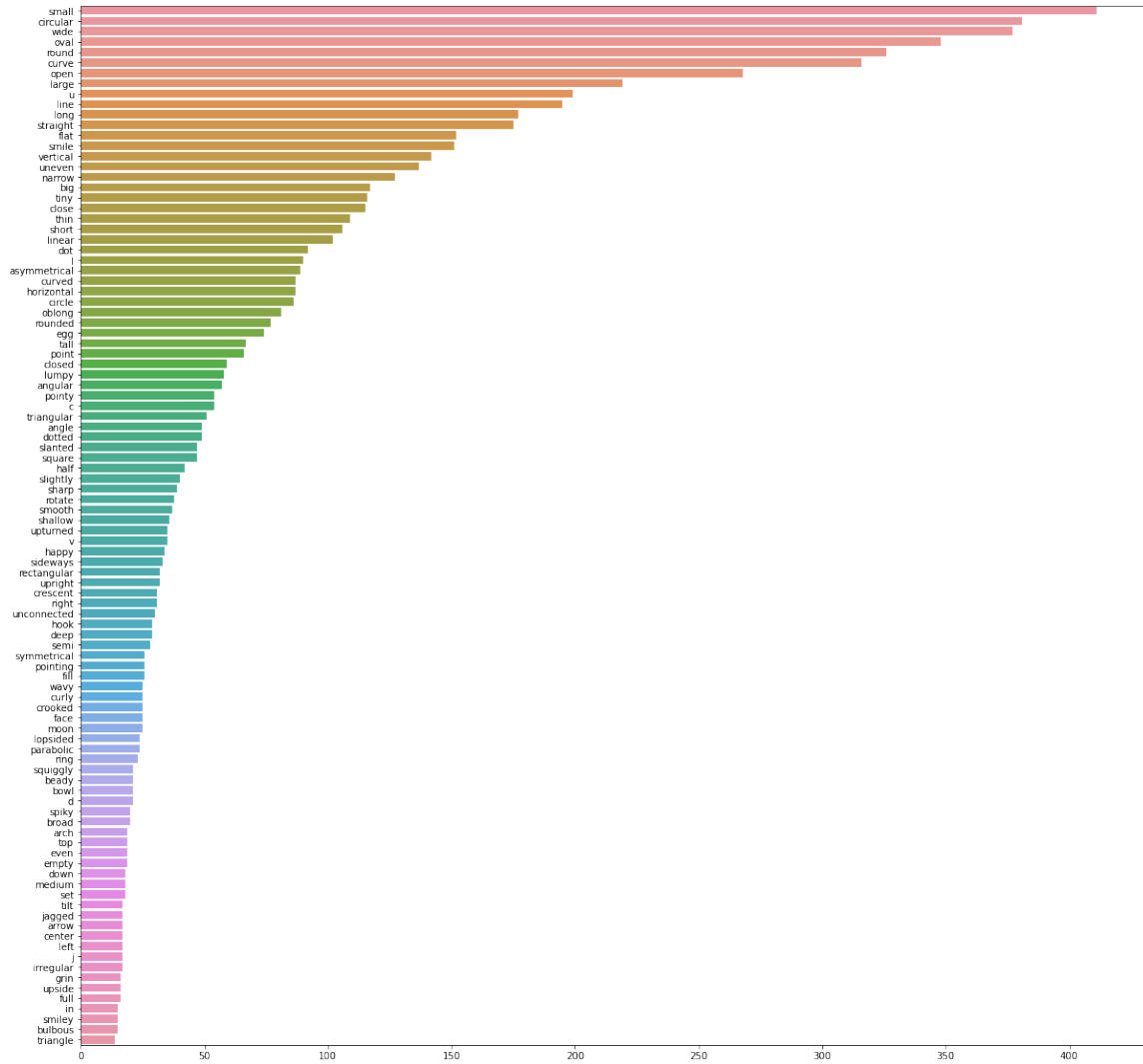Figure 3.14: The qualification questions.

Figure 3.15: Top 100 most frequent words in the dataset corpus.

Chapter 4

Modeling

Our goal is to build a collaborative drawing agent, through communication in language. In the Sketch-RNN work, the model can complete a partial sketch done by the users. In our case, we want to augment this collaborative creative process with language. A user can not only draw part of the sketch but also specify to the robot what kind of sketches they want to create: an angel with cloud-shaped wings or a face with angry-looking eyes. Learning sketch representations comes with its own challenges due to its abstractness, but how similar shapes are used in different context and how similar language is used in different context also makes learning sketch representation interesting, especially when most of the state of the art vision-language work focuses on RGB image data, we want to know how well SOTA methods generalize to the sketch domain. To advance towards a collaborative drawing agent, we choose sketch for its abstractness. The abstract nature of sketches brings both challenges and simplication of the problem. Challenges are about pretrained large vision-language models like CLIP are trained on RGB image data, simplification is about the fact that things like paintings with brushes bring in more manipulative challenges, so if we do want to study how language interacts with users or users interact with robots, it comes with its own research questions. Therefore, there are a couple of disparate directions that we can take this project. Simplication: it reduces the manipulation challenges with handling different art tools and techniques related to painting.

1. How well does current vision-language joint embeddings capture the abstract sketch representation? We are interested in this question because (1) CLIP is not trained with sketches and is trained mostly with RGB images; (2) humans are able to generalize concepts like small, large

We utilize CLIP to gain more insights into our dataset, beyond simple counting statistics. Given the nature of our dataset: a large variety of words but most words have very few occurrences, small number of images, text descriptions are contrastively collected by juxtaposing two images with opposite features, similar descriptions used for different purpose in different context. How does CLIP respond to this dataset, how well does CLIP embeddings align with our intuitions about these tasks? Specifically, for transferring the same word to be used in different context, or usage of words that are not the common meaning of words, how well can CLIP handle them, since it is trained on millions of images? Even though CLIP is not on the sketch domain, CLIP was trained on a large number of images on the internet, and there are GAN methods that have taken advantag eof CLIP embeddings to guide image generation: ClipDraw, StylCLIP, CLIP-NADA.

4.1   Task Definition

Given two sketches (s_1, s_2) and their part annotations (t_1, t_2), determine which sketch t_1 should pair with, similarly for t_2. During data collection, we implicitly juxtapose two sketches, chosen to be as distinct as possible using some heuristic, either from different clusters or whose cosine distance is large, so the process of annotating the two dissimilar sketches is like the annotators are choosing the pair up one annotation with another. Implicitly, the annotators is pairing s_1 with t_1 and s_2 with t_2, so we would regard the ground truth pairing to be (s_1, t_1) and (s_2, t_2). We want to see how CLIP does on this task, if it is the annotator for the task, would it be able to generate the same pairing. Define cosine similarity to be.

Given (s_1, s_2), we use CLIP image encoder (zero-shot/fine-tuned) f_v to extract visual features for the two sketches, $f_v(s_1) \in R^{512}$, and $f_v(s_2) \in R^{512}$. We then use the zero-shot/fine-tuned CLIP text encoder to extract the text features for the part descriptions, namely we fill in the template $t = $ [ADJ] [PART NAME], where [ADJ] is filled with the adjective phrases annotations, and [PART NAME] is the name of the part in the sketches. For angels, [PART NAME] is one of *halo*, *eyes*, *nose*, *mouth*, *body*, *outline of face*, *wings*; for face, [PART NAME] is one of *eyes*, *nose*, *mouth*, *hair*, *outline of face*. After filling in the above template, we obtain the part annotations for the two sketches t_1, t_2. We obtain embeddings for the part annotations by encoding them through CLIP text encoder f_t: $f_t(t_1) \in R^{512}$, and $f_t(t_2) \in R^{512}$. We then calculate cosine similarity between all four pairs of $(f_v(s_i), f_t(t_j))$, $i, j \in [2]$, where consine similarity between two vectors u, v is defined as $S_c(u, v) = \dfrac{u \cdot v}{\|u\| \|v\|}$. Therefore, given that our entire pipeline is f, $f(j) \in [2]$ output which of the

two sketches t_j will be paired with, and

$$f(j) = \max_i S_c(f_v(s_i), f_t(t_j)) \qquad i \in [2]$$

.

4.2   Metric

Given n pairs of two sketches and two part annotations, the same pairs that were provided by the annotators, we calculate an accuracy-like metric:

$$acc = \frac{\sum_{k=1}^{n} \sum_{j=1}^{2} \mathbb{1}(f(j) = j)}{2n}$$

4.3   Method

We utilized the `Python clip` package and load their pre-trained CLIP model, specifically the `ViT-B/32` variant, which uses the Vision Transformer (Dosovitskiy et al., 2020a) as the image encoder; `B` stands for *Base* model, and `32` stands for 32×32 input patch size. CLIP has two main parts: a text encoder and an image encoder, and both are transformer based. We will divide this section into explaining the two encoders. When introducing the text encoder, we will briefly introduce the encoder-decoder recurrent neural networks (RNN), which are the state-of-the-art method predating transformers, and their shortcomings in terms of inefficiency handling long sequences; we then introduce the self-attention mechanism and transformer models. In this way, we have better understanding of the advantages of modeling large corpus with transformer models. Transformers are first tested in natural language processing tasks and then adapted to vision tasks,

### 4.3.1   Text Encoder

Before Transformer: Recurrent Neural Networks   Traditionally, recurrent neural network's (RNN), such as long short-term memory (LSTM) and gated reccurent unit (GRU) recurrent network, are used in language modeling. The sequential nature of RNN's precluded them efficient parallelization and resulted in difficulty modeling longer sequences (Vaswani et al., 2017). Cho et al.

(2014) and Sutskever et al. (2014) introduced the framework of using neural networks, specifically RNNs, for language modeling. The framework works as follows: if given a sequence of T words $\mathbf{x} = (x_1, x_2, \ldots, x_T)$, an RNN encoder compute a sequence of hidden states $h_t = f(h_{t-1}, x_t)$, where h_{t-1} is the previous hidden state, and x_t is the current input. It then compresses all the T hidden states into one context vector $c = q(\{h_1, \ldots, h_T\})$, where q is some nonlinear function. Therefore, summarizing the input sequence into a representation is carried out in a sequential manner. The decoding process is sequential too: c and outputs from previous time steps are used to generate the output of the current step t, $y_t = g(c, h_t^{'}, y_1, \ldots, y_{t-1})$, where g is some nonlinear transformation, and $h_t^{'}$ is the hidden state in the decoder (Cho et al., 2014; Sutskever et al., 2014). In this way, the encoder-decoder RNN architecture models the output sentence sequence $\mathbf{y}$ as the product of conditionals of each time step: $\mathbf{y} = \prod_{t=1}^{T} p(y_t|c, y_1, \ldots, y_{t-1})$. During both encoding and decoding, the RNN architecture constrains the modeling to be sequential (Bahdanau et al., 2014). Moreover, the sequential nature of RNN limits the ability to model long-range dependencies among parts of the sentences, which is crucial in many cases.

RNN with Attention   The attention mechanism, the foudational building block of transformer, has been used with RNN's before. Introduced in Bahdanau et al. (2014), attention is used to alleviate the problem of compressing all information extracted from the input sequence in one fixed context vector c to compute the output at every time step. In the aforementioned $y_t = g(c, h_t^{'}, y_1, \ldots, y_{t-1})$, Bahdanau et al. (2014) altered the fixed c into an hidden state dependent c_t, and it is computed as the weighted sum of the encoder hidden states h_j: $c_t = \sum_{j=1}^{T} \alpha_{tj} h_j$. The weight α_{tj} represents how well h_j, the hidden state corresponding to the j-th word in the input, align with the decoder hidden state $h_{t-1}^{'}$ just before the current decoding step t. In this way, c_t summarize the input in an output-dependent way, making inputs that associate with y_t more closely contribute more to modeling $p(y_t|c_t, y_1, \ldots, y_{t-1})$.

Transformer: Attention is All You Need   The text encoder of CLIP is based on the transformer architecture introduced in Vaswani et al. (2017), and transformer relies only on the self-attention mechanism to compute a representation for the input sequence. In this way, it alleviates the computation efficiency and long-range dependencies problems witnessed in recurrent layers. Figure 4.1a and 4.2 are the same figures used in Vaswani et al. (2017) to illustrate the transformer architecture. In Figure 4.1a, Vaswani et al. (2017) gives an overview of the encoder architecture; tokenized texts go through an embedding layer, and the input embeddings are summed with learned position embeddings that inject information on order of the sequence. The input is computed using Byte Pair Encoding (BPE) with a $49,152$ vocabulary. As explained in Radford et al. (2019),
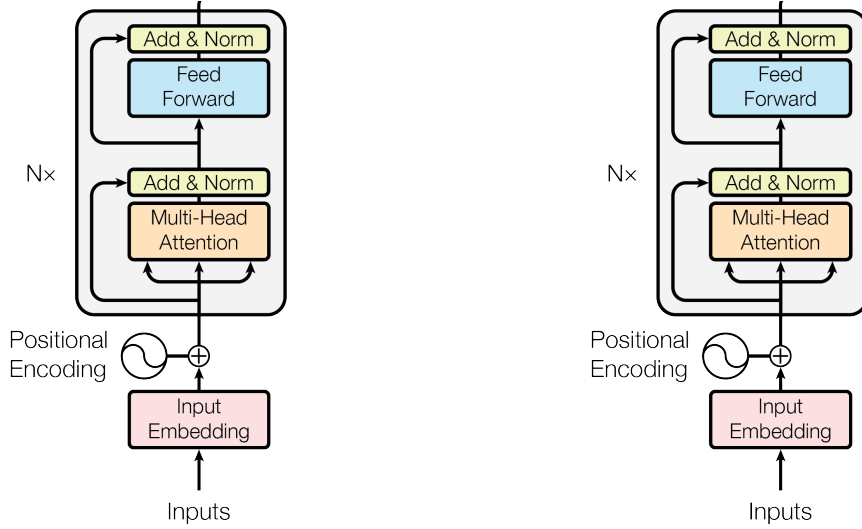
(a) Encoder architecture used in original Transformer paper (Vaswani et al., 2017).

(b) Encoder architecture of CLIP (Radford et al., 2021).

Figure 4.1: Transformer architecture.

BPE, a sub-word tokenization scheme, strikes a good balance between word-level and character-level word embeddings, since one works well with common words and the other with rare sequences. The input is then passed through stacked layers multi-head attention (MHA) mechanism followed by point-wise feed-forward networks (FFN). In the version of CLIP that we use, the text encoder is a 12-layer transformer, and the model dimenison is 512, $d_{model} = 512$, meaning that output of the initial embedding layers, MHA, FFN all have dimension 512. Compare to the formulation $LayerNorm(x + Sublayer(x))$ used in Vaswani et al. (2017) (Figure 4.1a), the CLIP text encoder uses $x + Sublayer(LayerNorm(x))$ (Figure 4.1b), where $Sublayer$ refers to either MHA or FFN, so each layer still contains residual connection and layer normalization, but the order is switched.

As illustrated in Figure 4.2b, given query, key, value matricies Q, K, V (in our case, all three matrices equal to the input text embeddings), transformer uses different linear projections to create multi-head attention, and Vaswani et al. (2017) explains the benefit of multi-head attention as allowing the model to attend simultaneously to multiple representation subspaces of the input.
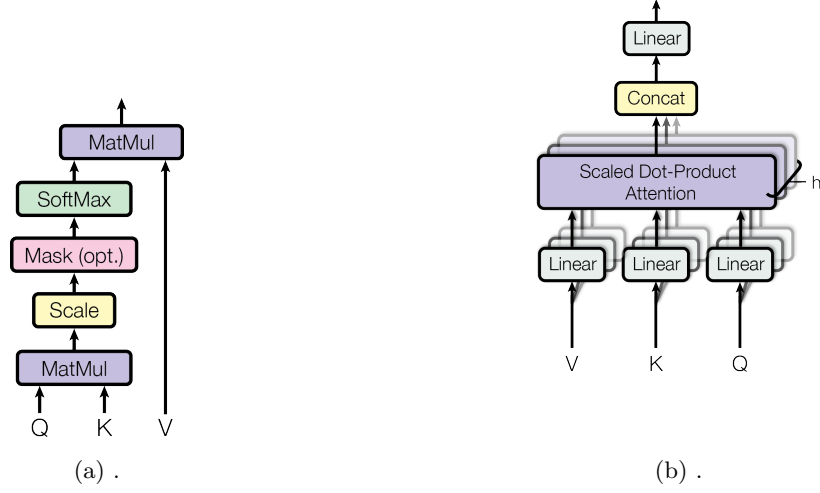
Figure 4.2: Screenshots of counterexamples used in first version of the requirements in Version 1.

$$
\begin{aligned}
MultiHead(Q, K, V) &= Concat(head_1, \ldots, head_h)W^O \\
head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \\
Attention(Q, K, V) &= softmax(\frac{QK^T}{\sqrt{d_k}})V
\end{aligned}
\tag{4.1}
$$

ViT-B/32 uses a version with $h = 8$ attention heads, so $W_i^Q, W_i^K \in \mathbb{R}^{d_{model} \times d_k}$ and $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, where $d_k = d_v = \frac{d_{model}}{h} = \frac{512}{8} = 64$. At the end of the multi-head attention mechanism, the weighted combination of values from each head is concatenated together and passed through a linear layer, represented here as $W^O \in \mathbb{R}^{(d_v \times h) \times d_v}$. CLIP uses the "Scaled Dot-Product Attention" in Vaswani et al. (2017), illustrated in details in Figure 4.2a. The dot products between query and key determine the weights that are used to sum the values; in this way, we have a contextualized representation; compared to convolutions that use static kernels, attention weights are dynamic.

4.3.2 Vision Transformer

The image encoder of CLIP uses Vision Transformer (ViT) introduced in Dosovitskiy et al. (2020b). The architecture of ViT is very similar to the original transformer introduced in Vaswani et al. (2017). In order to reuse the transformer model, ViT needs to first turn an image of size $H \times W \times C$, (H, W, C stands for image height, width, channel size, respectively), into a sequence of
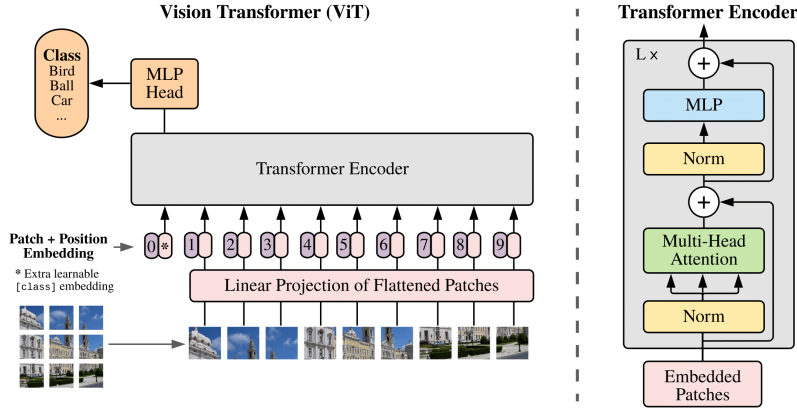
Figure 4.3: Vision Transformer (ViT) architecture (Dosovitskiy et al., 2020b).

"tokens", similar to the text input. To do so, Dosovitskiy et al. (2020b) reshapes the image to size $N \times (P^2 \cdot C)$, where N is the number of patches and P the patch size; the reshaped image can be seen as a sequence of N image tokens, each having a dimension of $P^2 \cdot C$. Each image token is then passed through a linear layer to be mapped to dimension D, similar to the model dimension d_{model} earlier. In the version of CLIP that we used, $H = W = 224$, $C = 3$, $P = 32$, and $D = 768$. As explained in Dosovitskiy et al. (2020b), before passing into the transformer, we also need to prepend a [class] token at the front the sequence, whose embedding at the last layer of the transformer will be used as the representation for the entire image. In this way, each image is represented as a sequence of $7 \times 7 + 1 = 50$ tokens, illustrated as purple boxes in Figure 4.3. The pink boxes that are right next to the patch embeddings represent position embeddings, with a similar function to encoder sequence order information as in the text transformer. For the CLIP image encoder, layer normalization is applied to the input before passing into the transformer and to the output at the last layer.

4.3.3   Pre-Training with Contrastive Objective and Natural Language Supervision

learning an open set of visual concepts from natural language natural language supervision compared to standard crowd-sourced labeling for image classification

It is difficult to predict the exact captions for an image, since there is usually a wide variety of text descriptions co-occuring with images, so instead CLIP turns to contrastive objectives and solves the problem of determining which text and image should be matched together. As shown in
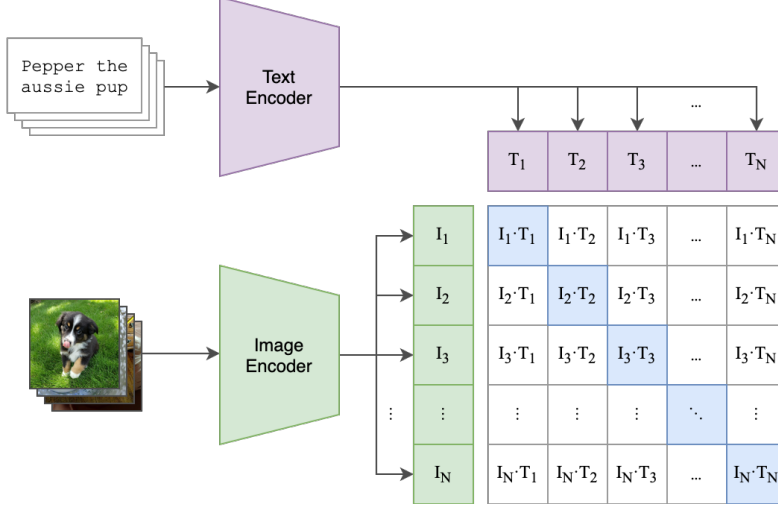
(1) Contrastive pre-training

Figure 4.4: CLIP uses contrastive pre-training instead of generative objective to learn joint vision-language embeddings.

Figure 4.4, during pre-training, for a batch of N (text,image) pairs, CLIP obtains N image features and N text features from the encoders. It then calculates the cosine similarities between each of the $N \times N$ pairings of the image and text features. The values are used as logit scores for calculating symmetric cross-entropy loss used to train the network.

CLIP is trained on 400 million (image, text) pairs collected form a variety of publicly available sources on the Internet.

The query list that is used to compile these (image,text) pairs is the union of (1) words that occur ≥ 100 times in English Wikipedia; (2) names of Wikipedia articles whose search volume is above certain threshold; (2) high pointwise mutual information (PMI) bi-grams, and (4) $117,000$ WordNet synsets, or sets of cognitive synonyms.

4.3.4   Image Pre-Processing

We use the data provided by SPG (Li et al., 2018), which provides JSON files of the Quick,Draw! sketches in vector format: each sketch is composed of a sequence of n strokes $S_i, i \in [n]$, and S_i is a sequence of vectors $(\delta x, \delta y, p, l)$. δx and δy are changes in the x, y coordinates with respect to the previous point; for the first point, it is with respect to $(25, 25)$. All points are assumed to be drawn

on a 256×256 canvas. $p = 1$ if the point is the last point in the current stroke, and $p = 0$ otherwise. The SPG dataset provides annotation for semantic segmentation of the sketches, so l is a number representing the semantically meaningful object part.

4.3.5   Text Pre-Processing

We used the `spacy` package to preprocess the text. `spacy` provides trained natural language processing pipeline and includes models for, for example, token-to-vector and part-of-speech tagging. We use the `en_core_web_sm` pipeline and its lemmatizer to reduce words to their basic forms. Moreover, we lower-case all words and remove punctuation, a list of which is provided by `Python string` package, `string.punctuation`. We also remove words like *shaped*, *sized*, *and*, *like*, since they act like stop words and do not provide additional visual descriptions of the sketches. Text descriptions are also tokenized by CLIP's tokenizer before passing into CLIP text encoder.

4.4   Loss Function

During training, for a given batch size b, we have b sketch-text pairs, $(s_k, t_k), k \in [b]$. We are essentially using classification over b classes to finetune CLIP, using cross-entropy loss. With clip, we obtain image logits X_v over the text descriptions and text logits X_t over the sketches. The ground-truth, for both image and text, is $Y_v = Y_t = \begin{bmatrix} 1 & 2 & \cdots & b \end{bmatrix}^T$

$$L(X_v, Y_v) = \frac{1}{b} \sum_{k=1}^{b} -\log \frac{\exp X_{vk,k}}{\sum_{c=1}^{b} \exp X_{vk,c}}$$

And similarly defined for (X_t, Y_t),

$$L(X_t, Y_t) = \frac{1}{b} \sum_{k=1}^{b} -\log \frac{\exp X_{tk,k}}{\sum_{c=1}^{b} \exp X_{tk,c}}$$

The final loss is defined as:

$$L = \frac{1}{2}(L(X_v, Y_v) + L(X_t, Y_t))$$

4.5 Data Augmentation

As mentioned above, our dataset has a small number of sketches: 572 face sketches and 787 angel sketches.

Chapter 5

Results & Analysis

5.1 Classification Experiments

	Face		Angel	
	Test	Dev	Test	Dev
zero-shot	0.54	0.55	0.56	0.57
finetuned on face	0.67	0.67	0.59	0.58
finetuned on face + angel	0.71	0.70	0.67	0.69

Table 5.1: Statistics of CLIP

Most pairs of words that people have used to differentiate the two sketches, most of the contrastive pairs of descriptions has decreased similarity. On average, in GLoVE, the distance between the contrastive words is 0.9. Compared to pre-trained CLIP, the average distance is 0.9. The two distance is roughly similar. However, word embedding calculated by CLIP fine-tuned on both face and angel category, the distance is 0.7. The average percentage of change is 14 percent increase in distance. Most contrastive words have moved further each other, resulting in the increase in accuracy.

We can calculate the cosine similarity between every pair of words in our corpus. Before fine-tuning, for each word w_i, it has a S_i of top-k most similar words. What can changes in S_i inform us of the dataset?

What does it mean to calculate the largest "mover" in our dataset? The words can only relative

46

to one another.

If the ultimate goal of using CLIP is to use it to guide the generation of each part, does the fine-tuned CLIP have this capacity?

Each caption introduces a visual concept.

Chapter 6

Future Work

Bibliography

Allado-McDowell, K., & Okojie, I. (2020). *Pharmako-ai.* Ignota.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). *Neural machine translation by jointly learning to align and translate.* arXiv. Retrieved from `https://arxiv.org/abs/1409.0473` doi: 10.48550/ARXIV.1409.0473

Bau, D., Strobelt, H., Peebles, W., Wulff, J., Zhou, B., Zhu, J., & Torralba, A. (2019). Semantic photo manipulation with a generative image prior. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, *38*(4).

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... Amodei, D. (2020). *Language models are few-shot learners.* arXiv. Retrieved from `https://arxiv.org/abs/2005.14165` doi: 10.48550/ARXIV.2005.14165

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). *Learning phrase representations using rnn encoder-decoder for statistical machine translation.* arXiv. Retrieved from `https://arxiv.org/abs/1406.1078` doi: 10.48550/ARXIV.1406.1078

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... Houlsby, N. (2020a). *An image is worth 16x16 words: Transformers for image recognition at scale.* arXiv. Retrieved from `https://arxiv.org/abs/2010.11929` doi: 10.48550/ARXIV.2010.11929

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... Houlsby, N. (2020b). *An image is worth 16x16 words: Transformers for image recognition at scale.* arXiv. Retrieved from `https://arxiv.org/abs/2010.11929` doi: 10.48550/ARXIV.2010.11929

Eitz, M., Hays, J., & Alexa, M. (2012). How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, *31*(4), 44:1–44:10.

Ge, S., Goswami, V., Zitnick, C. L., & Parikh, D. (2020). *Creative sketch generation.* arXiv. Retrieved from `https://arxiv.org/abs/2011.10039` doi: 10.48550/ARXIV.2011.10039

Ha, D., & Eck, D. (2017). *A neural representation of sketch drawings.*

Li, K., Pang, K., Song, J., Song, Y.-Z., Xiang, T., Hospedales, T. M., & Zhang, H. (2018). *Universal perceptual grouping.* arXiv. Retrieved from `https://arxiv.org/abs/1808.02312` doi: 10.48550/ARXIV.1808.02312

Qi, Y., & Tan, Z.-H. (2019). Sketchsegnet+: An end-to-end learning of rnn for multi-class sketch semantic segmentation. *IEEE Access*, *7*, 102717-102726. doi: 10.1109/ACCESS.2019.2929804

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., . . . Sutskever, I. (2021). *Learning transferable visual models from natural language supervision.* arXiv. Retrieved from `https://arxiv.org/abs/2103.00020` doi: 10.48550/ARXIV.2103.00020

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners..

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. (2016). *Generative adversarial text to image synthesis.* arXiv. Retrieved from `https://arxiv.org/abs/1605.05396` doi: 10.48550/ARXIV.1605.05396

Sangkloy, P., Burnell, N., Ham, C., & Hays, J. (2016, jul). The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Trans. Graph.*, *35*(4). Retrieved from `https://doi.org/10.1145/2897824.2925954` doi: 10.1145/2897824.2925954

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). *Sequence to sequence learning with neural networks.* arXiv. Retrieved from `https://arxiv.org/abs/1409.3215` doi: 10.48550/ARXIV.1409.3215

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). *Attention is all you need.* arXiv. Retrieved from `https://arxiv.org/abs/1706.03762` doi: 10.48550/ARXIV.1706.03762

Wah, C., Branson, S., Welinder, P., Perona, P., & Belongie, S. (2011). *The Caltech-UCSD Birds-200-2011 Dataset* (Tech. Rep. No. CNS-TR-2011-001). California Institute of Technology.

Xu, G., Kordjamshidi, P., & Chai, J. Y. (2021). *Zero-shot compositional concept learning.* arXiv. Retrieved from `https://arxiv.org/abs/2107.05176` doi: 10.48550/ARXIV.2107.05176

Xu, P., Hospedales, T. M., Yin, Q., Song, Y.-Z., Xiang, T., & Wang, L. (2020). *Deep learning for free-hand sketch: A survey.* arXiv. Retrieved from `https://arxiv.org/abs/2001.02600` doi: 10.48550/ARXIV.2001.02600

Xu, T., Zhang, P., Huang, Q., Zhang, H., Gan, Z., Huang, X., & He, X. (2017). *Attngan: Fine-grained text to image generation with attentional generative adversarial networks.* arXiv. Retrieved from `https://arxiv.org/abs/1711.10485` doi: 10.48550/ARXIV.1711.10485

Yu, Q., Liu, F., Song, Y.-Z., Xiang, T., Hospedales, T. M., & Loy, C. C. (2016). Sketch me that shoe. In *2016 ieee conference on computer vision and pattern recognition (cvpr)* (p. 799-807). doi: 10.1109/CVPR.2016.93

Yuan, S., Dai, A., Yan, Z., Guo, Z., Liu, R., & Chen, M. (2021). Sketchbird: Learning to generate bird sketches from text. In *2021 ieee/cvf international conference on computer vision workshops (iccvw)* (p. 2443-2452). doi: 10.1109/ICCVW54120.2021.00277