Sketch Recognition

Kovács István

1. Introduction

Being someone artistically underdeveloped, while playing Activity with my friends, I always dreaded to pick drawing, because neither I nor my teammates knew what I was trying to communicate to them. I thought I was the problem, but when I stumbled upon a game called Quick, Draw![1], I realized that my teammates were just bad at recognizing doodles. It didn't matter how ugly my drawing was, the computer always guessed it. I could feel like Michelangelo while I drew some squiggly lines, and the game would always guess right. I was mesmerized by it and so I wanted to create something similar. Fortunately the dataset used by Quick, Draw! is available for the public in various formats, one just has to issue two commands and will find everything needed at one's fingertips. My initial goal was to incorporate the timing data of the sketches into them to simulate the continuous guessing of Google's algorithm, and then pass them into a simple image classifier to see how much accuracy is sacrificed in order to guess continuously, if any. Unfortunately the data is too big for my resources, so I had to cut some corners to create a working program, but the results I got might be somewhat interesting nonetheless.

2. Related Work

DeepSketch[2] is the first successful implementation of a Deep Neural Network in sketch recognition. The authors used ConvNets as a basis for similarity search using k-Nearest Neighbors. At the time of their publication, this was the state of the art in sketch recognition. Its strengths include being able to extract medium and high level features, while it does not support guessing on the fly.

The paper called Enabling My Robot To Play Pictionary[3] describes a solution using a Recurrent Neural Network. It introduces a Gated Recurrent Unit based framework which helps in achieving state-of-the-art results. It supports recognizing sketches on the fly, enables users to play the popular party game Pictionary with their computer.

3. Proposed Approach

The data coming from the files contains pairs of coordinates in the range [0,255] and is grouped into strokes. The program iterates through the strokes and draws the image using a predefined color, then it gets scaled down to 32x32 and normalized from [0,255] into [-1,1].

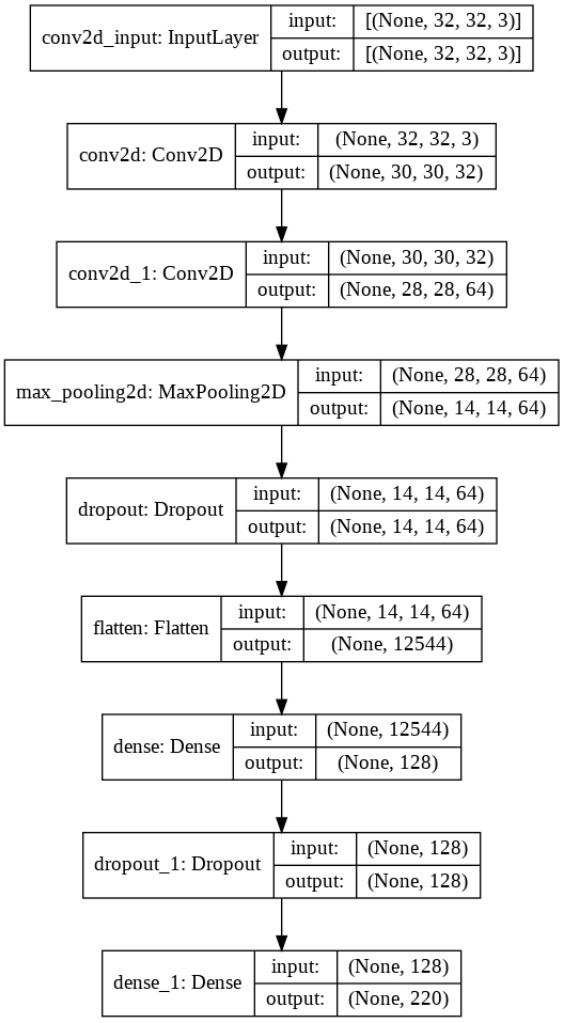

Figure 1. model architecture

The network used (figure 1) is a Convolutional Neural Network. It was chosen, because I wanted to compare parsing of static data to the on-the-fly recognition of the RNN used by the game. A shallow network is used, because of the limited number of samples per category that was decided to allow more categories.

As for evaluation metrics, only accuracy is used, be-

cause the data is balanced and the sketches, being made by humans, are not "fair", as in deliberately faulty drawings, drawings with 400 strokes(keep in mind that the game gives you 20 seconds to write an object). Categorical cross-entropy loss function and Adam optimizer are used.

4. Experimental Results

The dataset used is a subset of the binary version of the simplified Quick, Draw! dataset. It contains 100.000 - 200.000 images for 345 categories. The subset used contains 220 categories with 800 images each.

The relevant data of an image is the list of strokes, each stroke containing a list of x and a list of y coordinates, which are some points in the stroke the user drew. I opted for more categories rather than more images per category out of curiosity to see how much can be achieved with limited data. The training set contains 768 images, the rest is used for testing after the run. For benchmarking the application, another random sample of 800 images per category is used.

My network achieved a staggering 49.6% accuracy, which, compared to models like deepsketch, which has a 75.42% accuracy achieved on a dataset with 250 categories across 20.000 images, means that there's a lot of room for improvement. Other notable networks are sketchrnn, which is closely related to quickdraw, achieved 80% accuracy on the quickdraw dataset and also generates images. Another RNN, [3], achieved 85.1% on a database with 160 categories with 56 images each, and used data augmentation techniques.

5. Conclusions

This research did not prove that preprocessing data can simulate the timing information within a sketch, but I believe that it laid a foundation for further investigations in the area.

References

[1] Google. Quick, draw!

[2] Omar Seddati, Stéphane Dupont, and Mahmoudi Saïd. Deepsketch: Deep convolutional neural networks for sketch recognition and similarity search. volume 2015, pages 1–6, 06 2015.

[3] Ravi Kiran Sarvadevabhatla, Jogendra Kundu, and Radhakrishnan Venkatesh Babu. Enabling my robot to play pictionary : Recurrent neural networks for sketch recognition. *CoRR*, abs/1608.03369, 2016.