QuickDraw Sketch Recognition Using CNN and RNN

Fangqin Dai
fdai@stanford.edu

Xiaomeng Shen
xshen10@stanford.edu

Abstract

This article aims to apply and compare the performance of different kinds of deep neural networks on the Quick-Draw sketch dataset. Two types of neural networks, CNN and RNN are explored in this article. The final evaluation accuracy achieved were 93.2% and 89.8% respectively.

Compared to natural images, human sketches are sparse and lack of details, however a freehand sketch is composed of groups of sequences of strokes containing temporal ordering and grouping information, thus sketch recognition is a quite different problem than image classification. Most image classification methods including CNN discard such temporal information. When use CNN methods, we try to encode temporal information into images as much as possible; when use RNN methods, we use ordering information inherently.

1 Introduction

"Quick, Draw!" is an online game developed by Google that challenges players to draw a picture of a certain category, such as "alarm clock", "potato," etc. and then uses a neural network artificial intelligence to guess what the drawings represent. This game has generated over 1 billion drawings, and a subset of which will be used as training set to advance the model and increase its ability to guess with higher accuracy in the future. The work presented here will benefit the development on handwriting recognition and its robust applications in areas including OCR (Optical Character Recognition), ASR (Automatic Speech Recognition) and NLP (Natural Language Processing).

2 Related Work

Sketch is a simple and handy way to convey ideas that has existed since prehistoric times. While sketch recognition is straightforward for human, it could be a challenge for computer, due to the lack of rich texture details, inherent ambiguities, and its large shape variations.

There are currently three representations to process sketches for recognition:

- raster pixel sketch, where only stroke-covered area is marked as 1 while the rest is marked as zero. However, this approach does not allow CNN to assign different weights to strokes for better recognition.[10]
- vector sketch, where the input is a sequence of strokes in drawing order, such as described in sketchRNN[2]
- a single-branch attentive network architecture RNN-Rasterization-CNN (Sketch-R2CNN),[5] which takes advantages of both vector and raster representations of sketches during the learning process and is able to focus on adaptively learned important strokes, with an attention mechanism

With the advancement made in deep learning, lots of work has also been done on sketch recognition. Sketch-a-Net is specifically adapted for sketch images by using large kernels in convolutions to accommodate the sparsity of stroke pixels, and surpassed the human performance on TU-Berlin benchmark for the first time.[8][6]

ResNet50 has also been used to evaluate sketch recognition, as a CNN-variant method.[4]

Sketchmate as an RNN branch has demonstrated that temporal ordering in vector sketches can complement the other CNN branch for extracting more descriptive features.[7]

3 Data Prepossessing

3.1 Data format

The data in **train_simplified.zip** consists of 6 columns, key drawing information is stored as time-series strokes under "drawing", and each row contains one drawing. Examples of data structure are shown in figure 1 and figure 2. The data set is well-balanced with roughly 140k samples per class. We will use all 49 million samples for training and 112k samples as the test set.

Figure 1. Raw doodling data format

	countrycode	drawing	key_id	recognized	timestamp	word
0	US	[[[111, 148, 161, 175, 199, 218, 231, 236, 234...	5159910851477504	TRUE	2017-03-21 13:02:16.246170	alarm clock
1	SG	[[[91, 85, 49, 35, 32, 34, 41, 57, 88, 109, 13...	5959158429908992	TRUE	2017-03-17 04:32:30.717220	teddy-bear

Figure 2. Drawing array format

```
[
  [  // First stroke
    [x0, x1, x2, x3, ...],
    [y0, y1, y2, y3, ...],
    [t0, t1, t2, t3, ...]
  ],
  [  // Second stroke
    [x0, x1, x2, x3, ...],
    [y0, y1, y2, y3, ...],
    [t0, t1, t2, t3, ...]
  ],
  ... // Additional strokes
]
```

3.2 Remove whitespaces

Remove whitespaces from the `drawing` column in CSV files will save around 20% space for both memory and disk. After removing whitespaces the files size reduce from 23GB to 19GB.

3.3 Parse strokes with high performance

Many people are using `ast.literal_eval()` to parse the `drawing` column, actually it's much slower than `json.loads()`. We use both ways to parse the whole simplified dataset and store to a single HDF5 file, to compare their performance.

Table 1. Parsing methods comparison

Empty	no parse	ast.literal_eval()	json.loads()
Time	5087s	13575s	2781s
HDF5 File Size	18G	13G	13G

Table 1 shows that `json.loads()` is about 5x faster than `ast.literal_eval()`.

3.4 Generate images on the fly

The training dataset contains 49707579 images, if we draw them into 256x256 RGB images, they will use $49707579 \times 256 \times 256$ bytes, which is about 8.89TB ! Reading so many files on disk is definitely much slower than generating images on the fly during training.

3.5 Use cv2 to generate images

There are a lot of people using the `pillow` library to manipulate images, it's a very popular library, however it's 9x slower than cv2, so do use cv2 instead of pillow to generate images.

3.6 Split and shuffle the training data

The training data is too huge to load into memory, instead we split the into 100 parts, shuffle then and save each part in a HDF5 file. During training we load one part into memory in turn. To be more memory efficient we only use the `drawing` and `word` columns. Do remember to shuffle the data before saving to HDF5 files.

3.7 Encode as much information as possible in images

The strokes in `train_simplified.zip` contain temporal ordering and grouping information. We use various ways to draw images from strokes, all of them aim to **encode as much information as possible in images**. However there is no silver bullet so we use all methods to draw images and train them respectively.

Multiple resolutions are used, including 112x112, 128x128, 192x192, 224x224, 256x256.

3.8 Prepare sequential data to RNN

- Running Example: When possible, use a running example throughout the paper. It can be introduced either as a subsection at the end of the Introduction, or its own Section 2 or 3 (depending on Related Work).

- Preliminaries: This section, which follows the Introduction and possibly Related Work and/or Running Example, sets up notation and terminology that is not part of the technical contribution. One important function of this section is to delineate material that's not original but is needed for the paper. Be concise – remember Guideline 1.

- Content: The meat of the paper includes algorithms, system descriptions, new language constructs, analyses, etc. Whenever possible use a "top-down" description: readers should be able to see where the material is going, and they should be able to skip ahead and still get the idea.

4 Experiments

4.1 Convolutional Neural Networks

Evaluation metric used in the project is Mean Average Precision @ 3 (MAP@3).

4.1.1 Simple CNN

We build a simple CNN network constructed with layers described in table 2. The number of filters increases with depth. Small stride of (1,1) is used after first convolutional layer to ensure capturing most information. Large filter size is chosen given the simplicity of doodling image. For pooling layer, we used overlap pooling with 3x3 pooling size and stride of 2.[1]

With this simple CNN network, after trained on the entire training dataset for one epoch, it achieves 0.82 top1 accuracy and top3 accuracy 0.85.

Table 2. The architecture of shallow CNN

type	filter Size	filter num	stride	pad
Conv1+ReLU	7x7	16	(2,2)	same
Conv2+ReLU	7x7	32	(1,1)	same
Conv3+ReLU	7x7	48	(1,1)	same
Maxpool	3x3		(2,2)	same
Conv4+ReLU	3x3	64	(1,1)	same
Conv5+ReLU	3x3	96	(1,1)	same
Maxpool	3x3		(2,2)	same
Conv6+ReLU	3x3	128	(2,2)	same
Conv7+ReLU	3x3	256	(1,1)	same
Conv8+ReLU	3x3	512	(1,1)	same
Maxpool	3x3		(2,2)	same
Flatten+FC				

This simple CNN is shallow therefore it's fast to train. We use it to test the whole training pipeline and make sure all code work smoothly together.

4.1.2 MobileNet and MobileNetV2

MobileNet is a fast and memory efficient CNN network[3], we use it as a starting point for this Kaggle contest. First we tried 64x64 grayscale images with static color, then we adopted dynamic color according to the ordering of strokes, table 3 shows that dynamic color can improve the accuracy significantly, which is expected because it encode temporal information into images.

We trained MobileNet and MobileNetV2 [9] with different resolutions and batch sizes, and we don't see any improvement from MobileNetV2 and it's even slower to train.

Table 3. Static color VS. dynamic color

Model	Image Size	Color	Batch Size	Epochs	Time (h)	Top3 Acc
MobileNet	64	black	680	1	3.6	0.875
MobileNet	64	dynamic	680	1	3.2	0.896

Table 4. MobileNet and MobileNetV2

Model	Image Size	Color	Batch Size	Epochs	Time (h)	Top3 Acc
MobileNet	64	dynamic	680	3	7.43	0.915
MobileNetV2	64	dynamic	680	3	9.57	0.90

4.1.3 ResNet

ResNet has been the de-facto standard in image classification, so we applied it to the sketch recognition problem. We trained ResNet50 with different resolutions and different colors.

Table **??** shows the results of ResNet50 trained on 128x128 images with various colors, their differences are very close so there is no significant best dynamic color mechanism, in the end we use all colors including one static color and two dynamic colors.

Table 5. ResNet50 128x128 with various colors

Image Size	color	Batch Size	Epochs	Time (h)	Top3 Acc
128	static	260	1	22.43	0.892
128	color1	260	1	22.39	0.912
128	color2	260	1	22.41	0.919

Table 6shows that the larger resolution the better accuracy, which is reasonable.

4.2 Recurrent Neural Networks

The strokes are sequential data so it's worthy to try sequence models. All RNN models seem not good than CNN model in table 8.

5 Conclusions and Future Improvements

In general CNNs are better than RNNs, "CNN+RNN" architecture is the worst and "RNN+CNN" shoud be the best and we haven't seen it yet.

In order to ac hive higher accuracy, the "RNN+CNN" architecture worth exploring, and ensemble definitely can improve the accuracy by 1 percent at least.

Table 6. ResNet50 with different resolutions

Image Size	color	Batch Size	Epochs	Time (h)	Top3 Acc
64	static	1031	1	6.61	0.892
112	static	327	1	18.53	0.912
128	static	260	1	22.41	0.915
192	static	116	1	54.65	0.914
224	static	81	1	74.46	0.919
256	static	64	1	97.28	0.923

Table 7. Models Top3 Accuracy.

Model	Image Size	Batch Size	Epochs	Time (h)	Top3 Acc
MobileNet	64	680	1	3.2	0.909
MobileNetV2	64	680	1	3.5	0.868
ResNet50	64	1031	1	6.47	0.893
MobileNet	128	680	1	21.8	0.914
ResNet50	128	260	4	81.16	0.932

References

[1] I. S. A. Krizhevsky and G. E. Hinton. Imagenet classification with deep con- volutional neural networks. (25 10901098), 2017.

[2] D. Ha and D. Eck. A neural representation of sketch draw- ings. 2018.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[4] S. R. J. S. Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition. *In Proc. IEEE CVPR*.

[5] Y. Z. Q. S. H. F. C.-L. T. Lei Li, Changqing Zou. Sketch-r2cnn: An attentive network for vector sketch recognition. November.

[6] J. H. M. Eitz and M. Alexa. How do humans sketch objects? *ACM TOG*.

[7] T. Y. K. P. Y.-Z. S.-T. X. T. M. H. Z. M. P. Xu, Y. Huang and J. Guo. Sketchmate: Deep hashing for million-scale human sketch retrieval. *In Proc. IEEE CVPR*.

[8] F. L. Y.-Z. S.-T. X. Q. Yu, Y. Yang and T. M. Hospedales. Sketch-a-net: A deep neural network that beats humans. *IJCV*.

[9] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[10] R. G. Schneider and T. Tuytelaars. Sketch classification and classification-driven analysis using fisher vectors. *ACM TOG*, 33(6):174:1 – 174:9.

Table 8. RNNs

Model	Batch Size	Epochs	Time (h)	Top3 Acc
Conv1D+LSTM	432	1	8.43	0.873
ConvLSTM2D_Plain	277	1	25.73	0.885
ConvSTM2D_MobileNet	320	1	37.99	0.898