Sketchy - A Sketch Recognition System

Vasishtha Sriram Jayapati
University of Massachusetts Amherst
Amherst, MA, USA
vjayapati[at]cs[dot]umass[dot]edu

Ajay Venkitaraman
University of Massachusetts Amherst
Amherst, MA, USA
avenkitarama[at]cs[dot]umass[dot]edu

Abstract

Freehand sketch recognition is not a fully explored problem, as compared to other related problems such as image recognition since sketch recognition depends on extracting abstract features, as against images which are fully representable using the standard algorithms that we use. In this project we set out to compare the augmented forms of Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) on the Quick, Draw! dataset and evaluate the performance of both with respect to accuracy and training time. Our results show that RNN outperforms CNN in terms of accuracy by about 10% with a substantial trade-off in the training time.

1. Introduction

Freehand sketches have been used by humans to depict objects and life-forms since a very long time, and they are very different from the more elaborate paintings that we see around us. Recognizing these sketches is a different challenge from that of recognizing images, since images are analyzed pixel-by-pixel and algorithms have become adept at this, whereas when it comes to analyzing sketches, what matters more is the understanding of abstract features that help humans know that a sketch depicts a dog, or a human, for example.

For this task, a considerable amount of research is going on, and at the forefront of this research is the use of different neural networks for recognizing hand drawn sketches. Our system consolidates and builds upon two of these techniques, viz. Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) and we also evaluate the performance of each for the data we have chosen. Specifically, we use the RNN model based out of the work of Ha and Eck [4] for our system. The CNN model used in our project is derived from the ideas in [17, 15, 19]. The dataset that we will be using is the Quick, Draw! Dataset [9]. This dataset is a collection of over 50 million drawings across more than 300 categories. We will also be using the Simple Vector Drawing Datasets (SVD Dataset) from [5] for training our RNN model.

2. Related Work

One of the major impediments in research in the area of sketch recognition is the lack of availability of large enough datasets for training models, probably with the exception of the MNIST dataset [13]. In later times, the Sketch dataset [3] was used to explore feature extraction techniques on vector sketches. Subsequently, researchers from Georgia Institute of Technology developed the Sketchy database [14], providing a substantially larger dataset along with pixel images of the classes of sketches which can be used for baseline testing of the sketch recognition model. The Quick, Draw! Dataset [9] that we're using is probably the first large scale dataset that caters to sketch recognition and also offers a fun game for the people to play, in turn contributing more data to the dataset [10].

Neural networks based approaches for sketch recognition has been going on for a few years now, but most of the research has been focused on learning pixel images [11, 18, 6]. Only recently have inroads been made to tackle the problem of sketch recognition using neural networks [17, 4], and these are standalone research topics that have focused on one specific technique for sketch recognition. To our knowledge, there hasn't been any major work that has set out to implement and compare the different approaches for a particular dataset. Thus, our topic of research seems a good extension on the previously explored areas.

3. Architecture

The architecture of the RNN module of our system is based on the one used by Ha and Eck in [4] and is shown in Figure 1. It is a Sequence-to-Sequence Variational Autoencoder (VAE), akin to the one described in [1, 7]. The encoder is a bidirectional RNN as the one in [16]. It takes a sketch as the input to the model and outputs a vector of size N_z. We feed the sketch sequence S as the input and the same sequence in reverse order, $S_{reverse}$ to two encoding

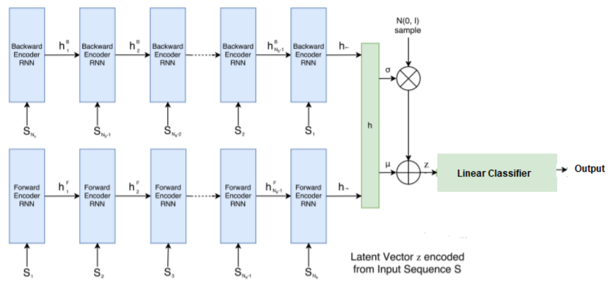

Figure 1: Schematic diagram of the RNN architecture.

RNNs and get two output states, which we concatenate to form h as the output state.

$$h_{\rightarrow} = encode_{\rightarrow}(S), h_{\leftarrow} = encode_{\leftarrow}(S_{reverse}), h = [h_{\rightarrow}; h_{\leftarrow}]$$

This final state h is projected onto two vectors μ and σ, each of size N_z. These μ and σ are used to output a latent vector, $z \in \mathbb{R}^{N_z}$ [4]. This latent vector z is passed through a linear classifier to obtain the output classes.

$$\mu = W_\mu h + b_\mu, \quad z = \mu + \sigma \mathcal{N}(0, I)$$

For the CNN module, we derive inspiration from [17, 15, 19]. These works build upon the well known architectures of ImageNet [8] and LeNet [12]. ImageNet consists of an 8-layer network with 5 convolutional layers and 3 fully connected layers. The output of the last layer is connected to a 1000-way softmax, and produces a distribution of the 1000 class labels. LeNet is a 7-layer network with 2 convolutional layers, 2 subsampling layers, 2 fully connected layers and a final Gaussian connected layer with 10 output classes [15].

4. Implementation Details

The RNN module of our system is based on the Sketch-RNN implementation by Magenta in TensorFlow by Ha and Eck [4]. The model cited above used a generative RNN module to draw sketches, whereas our system classifies images into their appropriate categories. We thus had to modify the loss function to softmax cross-entropy loss. Our implementation does not include the decoder part of the architecture mentioned in [4].

The CNN module as described above is implemented using the open source library Keras [2]. The specifics of the CNN model that we have used are given in the Table 1.

Ind	Type	Filter	Filter	Stride	Pad
1	Conv	3x3	64	1	1
2	ReLU	-	-	-	-
3	MaxPool	3x3	-	3	0
4	Conv	3x3	128	1	1
5	ReLU	-	-	-	-
6	MaxPool	3x3	-	3	0
7	Conv	3x3	64	1	1
8	ReLU	-	-	-	-
9	MaxPool	3x3	-	3	0
10	Fully Connected	-	128	-	-
11	ReLU	-	-	-	-
12	Fully Connected	-	10	-	-
13	SoftMax	-	-	-	-

Table 1: Table showing the specifics of the CNN model used

5. Experiments and Evaluation

5.1. Dataset

The dataset used for the project is the publicly available Quick, Draw! [9] dataset, which consists of over 50 million sketches in across 345 categories (or classes as used interchangeably in this section). This dataset was created by volunteers playing the Quick, Draw! game [10]. The sketches are timestamped vectors, tagged with metadata including what the volunteer was asked to draw and the country in which the player was located. We have chosen 10 classes from the preprocessed dataset in .npz format (for the RNN module) and in .npy format (for the CNN module). Each class consists of a training set of 70K samples, in addition to 2.5K samples each for validation and test sets [4].

The .npz files contain sketches which are a list of points, where each point is a vector comprising of the following five elements: (Δx, Δy, p_1, p_2, p_3). Here, the first two elements represent the offset distance in the x and y directions of the pen drawing the sketch from the previous point. The last three elements represent a binary one-hot vector of three possible states viz. the first pen state, p_1 which denotes that the pen is on the paper, the second pen state, p_2 denotes that the pen is off the paper. The final pen state, p_3 denotes that the drawing has been completed [4].

For the .npy files used for the CNN module the sketches are transformed into 28x28 grayscale bitmap in numpy .npy format. np.load() was used to load these files before passing them as the input to the CNN model.

5.2. Evaluation Results

The data from the ten classes namely cat, dog, bear, airplane, ant, banana, bench, book, bottle cap and bread was sampled from the dataset to obtain the train, validation and test data. This train data was then used to train both the RNN and the CNN models. The classification accuracy obtained using CNN on various splits of the test data using these models are shown in the Table 2.

Test to Train ratio	Accuracy (%)
0.25	85.27
0.33	84.89
0.5	84.67
Mean Accuracy	84.94

Table 2: Accuracy of the CNN model on different data splits

Dataset	Accuracy (%)
Quick, Draw!	88.34
SVD Dataset	93.27

Table 3: Table showing the accuracy of the RNN model for the two datasets

Model	Accuracy (%)
RNN	93.41
CNN	84.94

Table 4: Comparison of accuracy obtained by CNN and RNN models

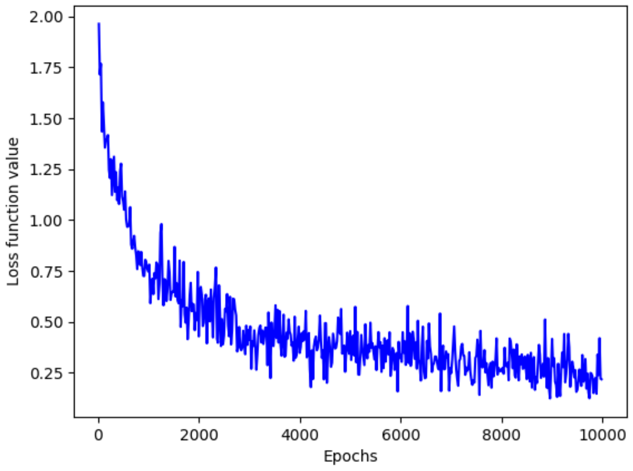

Figure 2: Plot of Loss Function vs Epochs

The RNN model was trained for several epochs and the value of the decreasing loss function with respect to the number of epochs was plotted as shown in Figure 2. Figure 3 shows the time taken for the RNN model for each epoch, which varied between 4 to 10 seconds.

For evaluating our RNN model, we have used all the three classes from [5] and the cat, dog and bear classes from [9] and the individual accuracies from the respective datasets are shown in Table 3. The RNN model was also evaluated on the aforementioned ten classes from [9] to get an overall comparison of the performance of CNN and RNN models. Table 4 shows the accuracy values obtained. All these evaluations were performed on Google Colab using a Tesla T4 GPU with 14.73 GB memory, and a clock rate of 1.59 GHz.

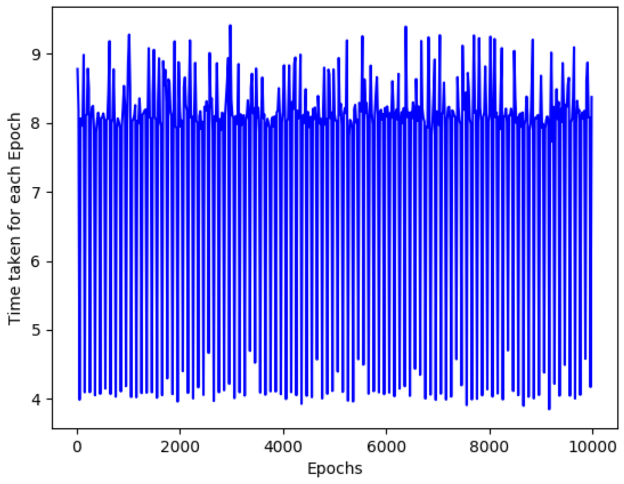

Figure 3: Plot of time taken (in seconds) for each epochs

6. Conclusion and Future Scope

In this project, we had an objective to compare the different state-of-the-art techniques for sketch recognition. Our results show that RNN outperforms CNN with respect to accuracy in all the datasets that we used. It is noteworthy that RNN also took far longer to train on the datasets (we didn't account the runtime for CNN since the model was much faster, in the order of around 10-20 seconds).

In the future, we plan to explore more into different CNN architectures. Also, due to memory constraints, we used only a fraction of the entire dataset and in the future we can look at the scalability of these techniques when trained over the entire dataset. Suggestions in [17] can also be used to incorporate an ensemble model consisting of native Machine Learning models such as kNN and XGBoost for detecting similar sketches.

References

[1] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. 2016.

[2] Franois Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[3] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Trans. Graph.*, 31(4):44–1, 2012.

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

[5] Hardmaru. Simple Vector Drawing Datasets. https://github.com/hardmaru/sketch-rnn-datasets, 2018.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Google Creative Lab. Quick, Draw! Dataset. https://github.com/googlecreativelab/quickdraw-dataset, May 2017.

[10] Google Creative Lab. Quick, Draw! Game. https://quickdraw.withgoogle.com/, May 2017.

[11] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[12] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] Yann LeCun, Corina Cortes, and Christopher J. C. Burges. The MNIST Database of Handwritten Digits. https://http://yann.lecun.com/exdb/mnist/, 1998.

[14] Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy database: learning to retrieve badly drawn bunnies. *ACM Transactions on Graphics (TOG)*, 35(4):119, 2016.

[15] Ravi Kiran Sarvadevabhatla and R Venkatesh Babu. Freehand sketch recognition using deep features. *arXiv preprint arXiv:1502.00254*, 2015.

[16] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[17] Omar Seddati, Stephane Dupont, and Saïd Mahmoudi. Deepsketch: deep convolutional neural networks for sketch recognition and similarity search. In *2015 13th International Workshop on Content-Based Multimedia Indexing (CBMI)*, pages 1–6. IEEE, 2015.

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[19] Yongxin Yang and Timothy M Hospedales. Deep neural networks for sketch recognition. *arXiv preprint arXiv:1501.07873*, 1(2):3, 2015.