Big Data Analytics Final Project Report

Topic: Nice Drawing - A tool that reproduces similar looking line drawings from sketches or photos

Xueyao Li, Yiyi Zhang
The Department of Electrical Engineering
Columbia University
xl2719@columbia.edu, yz3280@columbia.edu

Abstract—This report demonstrates our solution for the final project of E6893 Big Data Analytics. In this project, by utilizing tools including Google Cloud, Apache Spark, TensorFlow, Jupyter Notebook, Observable Notebook, we built a classification engine for doodle drawings of 15 animal categories and constructed a web-based application that recognizes the input of a sketched drawing or a photograph and returns an output of thousands similar looking line drawings.

I. Introduction

This project was inspired by Google's experiments on "Quick, Draw!" [1], an online game where the players are asked to draw objects belonging to a particular object class in less than 20 seconds, and AutoDraw [2], a drawing tool that pairs machine learning with drawings from artists to help users create drawings easier. Google creates the world's largest doodling dataset with the game "Quick, Draw!" and has made it publicly available [3]. We would like to leverage the dataset and be part of the community of exploring the application of the state-of-the-art technologies towards visual art and communication. Since most people perceive the world via their visual sense at the first place, we can see that drawing could help break down the languages barriers between people from different nations or people who do not have the ability to speak or write. We believe that by making the drawing experience easier, faster and more fun to the general public, it could become a new way of communicating with others on daily basis in the future, just as speaking and writing nowadays. To professional artists and designers, on the other hand, we would like to assist their creative process and help them expand imaginations as well. Therefore, we planned to create a tool that takes the input of a sketched drawing or a photograph and returns an output of thousands of similar looking line drawings. To work towards this goal, one of the first challenges is to correctly recognize the sketched drawings or photographs uploaded by people.

II. Related Works

The drawings classification shares some similarity with the handwritten digit recognition on the MNIST dataset [4], although it is considered to be more challenging in many aspects which will be discussed later in the Dataset section. [5] and [6] are two of the most cited papers related to the classification of the handwritten digit, and the best error rate has reached 0.21%. Regarding works related to the doodle drawings, sketch-rnn [7, 8] is a recurrent neural network model trained on the Quick, Draw! dataset and it can come up with many different possible ways to continue drawing the object based on where the users left off, while it requires to specify a particular object in advance. As the Quick, Draw! dataset became publicly available, there are emerging works related to the doodle classification, for instance, the TensorFlow API [9] and Kaggle's Competition sponsored by Google's AI [10]. With respect to the image recognition and object detection, there are much more related works, ImageNet Large Scale Visual Recognition Challenge [11] for instance, and it is still one of the major topics people try to improve on as of today.

III. System Overview

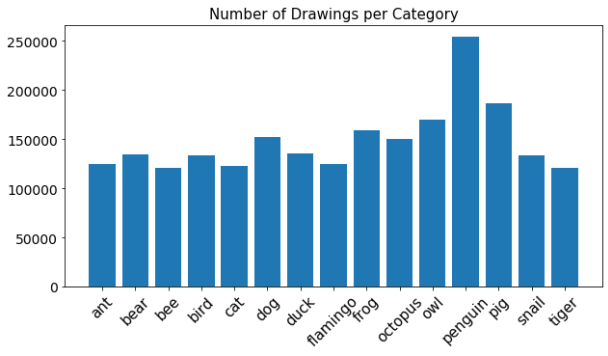
A. Dataset

The Quick, Draw! dataset consists of 50 million drawings across 345 categories, including animals, fruits, vehicles, and others. In this project, we chose 15 animal categories, (i.e., ant, bee, bear, bird, cat, dog, duck, flamingo, frog, octopus, owl, penguin, pig, snail, and tiger) from the whole dataset and aimed to build a classification engine for them. We used the dataset that had already been preprocessed to a uniform 28×28 pixel image size from the raw data that also includes the timestamps and the country code. Compared with the MNIST dataset, the "Quick, Draw!" data is much more challenging in many ways. First, the variability within each category is much bigger as there are many more ways to draw an animal than to write a specific number. Second, given the nature of the game design that the players are told to draw an object within 20 seconds while a neural network is predicting in real time and the drawing is stopped once the prediction is correct, the quality of the drawings is not guaranteed and the similarity between different categories is much greater, for instance, bird vs. duck and cat vs. tiger. *Figure 1* shows a sample of the drawings from each category, and we can see that there are lots of variability within a single category as well as many similarities across

different categories. *Figure 2* shows the number of drawings from each of the 15 animal categories. As the data is unbalanced, we decided to use 100K samples from each category, of which 80K as training set and 20K as the test set. The total size of the data we used for classification is around 1.85G.

Figure 1: Visualizing a sample of data from each category

Figure 2: Histogram of the number of drawings in each category



B. Pipeline

For the classification of doodle drawings, we decided to implement and compare two classification algorithms, i.e. Logistic Regression and Convolutional Neural Network (CNN), and then select the one with better performance to further improve on. The doodle classification would be our main focus for this project. For the classification of photograph input, we decided to leverage the TensorFlow Image Recognition and Object Detection API [12, 13]. We then would build a web-based interactive application that allows users to upload a sketched drawing or a photograph as input and returns a t-SNE [14, 15] map to visualize all drawings of the classified categories from the Quick, Draw! dataset.

IV. Algorithm

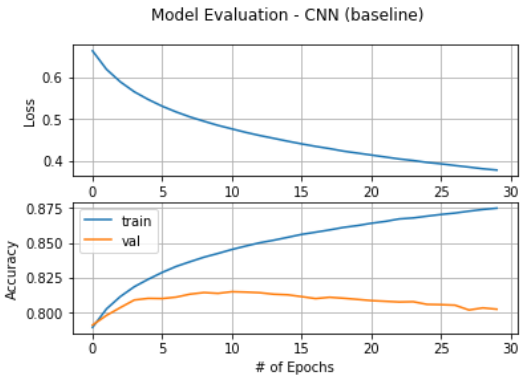
A. Basic Architecture

We implemented Logistic Regression with Apache Spark MLlib [16] and Convolutional Neural Network (CNN) with Keras [17]. The CNN was with the following seven layers:

1. Convolutional layer with 32 feature maps of size 3×3.
2. Pooling layer taking the max over 2×2 patches.
3. Convolutional layer with 64 feature maps of size 3×3.
4. Pooling layer taking the max over 2×2 patches.
5. Flatten layers.
6. Fully connected layers with neurons and rectifier activation.
7. Output layers.

The performance of CNN classifier with a different number of epochs is shown in *Figure 3.* As presented, the accuracy of the training set increased with an increase in the number of epochs while the accuracy of the validation set started to drop after 10 epochs. To prevent the model from overfitting, the number of epochs was chosen to be 10.

Figure 3: Evaluation of baseline CNN



The Logistic Regression and CNN provided us with an accuracy of 43.17% and 81.56% respectively. As we had expected, the CNN had better performance than the Logistic Regression while it required reasonably longer computing time. Given their performance, we decided to select the CNN classifier as our baseline model.

Figure 4: Confusion matrix of the baseline model performance

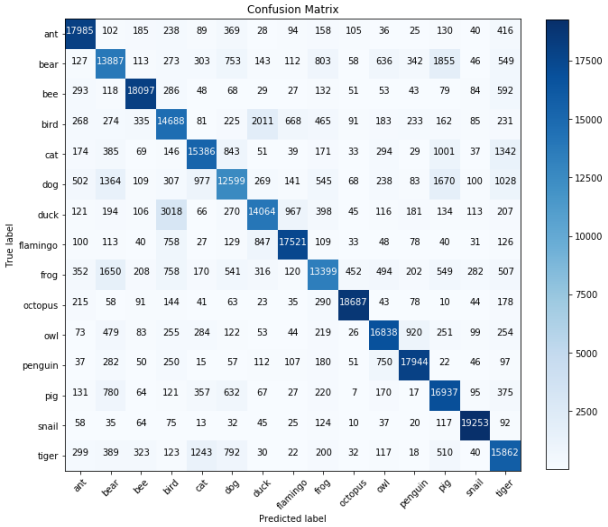


Figure 5a: Visualizing bird vs. duck

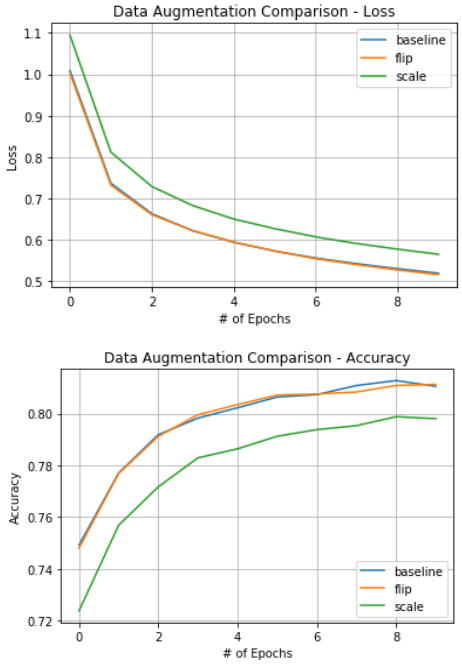
Figure 5b: Visualizing bear vs. dog/frog/pig

We constructed a confusion matrix to visualize the result and tried to understand which category had the best or worst predictions for which category. From the confusion matrix shown in *Figure 4*, we can see that snail and octopus had the best predictions while bear and bird had the worst predictions. This makes sense if thinking about the uniqueness of their characteristics compared to other animals in the categories. For example, other animals in the categories do not have any similar shapes as the shell of the snail or have as many arms and legs as the octopus, and therefore it would be easier to distinguish such categories from others. The shape of the bird or bear, however, has many similarities with that of other categories when coming across the doodle drawing, for instance, bird vs. duck (as shown in *Figure 5a*) or bear vs. dog/frog/pig (as shown in Figure 5b).

B. Data Augmentation

As the quality of the input data is significant to the performance of the model while there are many challenges presented in the dataset itself as discussed in earlier sections, in order to further improve the performance, we first tried to enhance the data [18] by randomly flipping or scaling the drawings of the training set. As shown in *Figure 6*, the validation accuracy did not improve with either of the methods. This may be the result that the dataset we used for the baseline was already big enough and given the nature of the drawings any further enhancements of existing data would not help much. Therefore, we decided not to apply any data augmentation techniques to the existing dataset.

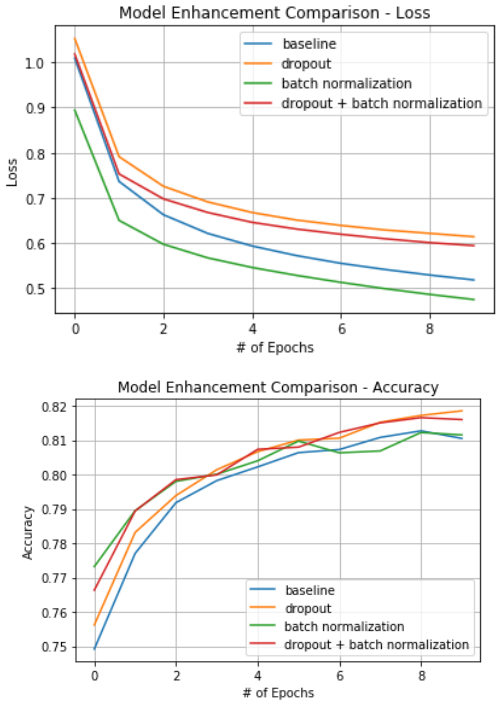
Figure 6: Data augmentation comparison



C. Model Enhancements

Next, we tried to improve the performance by enhancing the model using some general approaches such as dropout [19] and batch normalization [20]. As shown in *Figure 7*, by utilizing the batch normalization, the loss significantly decreased while the accuracy of the validation set was quite unstable. By further adding in dropout layers to prevent it from overfitting, the result became more promising. The test accuracy improved to 81.69% by adding both the dropout and batch normalization. Note that two different dropout rates (i.e., 0.25 and 0.5) were tested at different positions (i.e., after the second pooling layer or after the first dense layer or both). A dropout rate of 0.5 after the second pooling layer showed the best performance result.

Figure 7: Model enhancements comparison



V. Experiment Results

The prediction accuracy of our final model was 81.69%. We constructed the confusion matrix, as shown in *Figure 9*, to visualize and better understand the result as we did for the baseline model. *Figure 10* shows a few predictions that were classified incorrectly. There certainly are rooms for improvement since although there are many unconventional drawings, the humans may still be able to correctly recognize them or at least give a more accurate probability distribution. One of the ways to further improve the performance is to build a deeper convolutional network by adding more layers [21, 22].

Figure 9: Confusion matrix of the final model performance

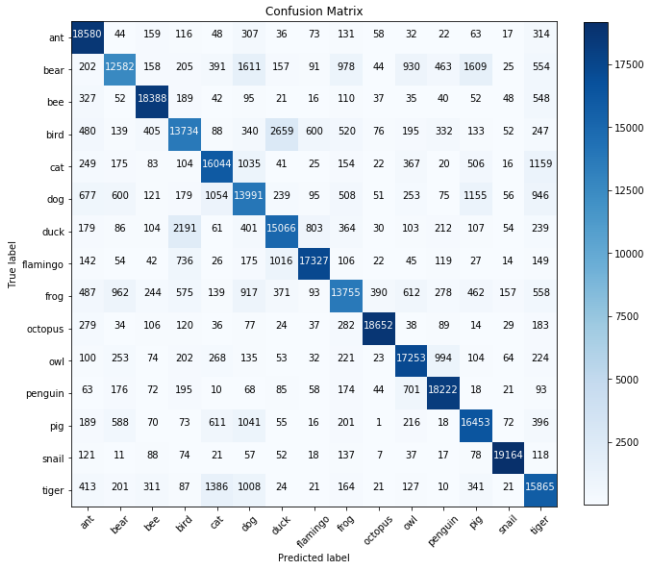
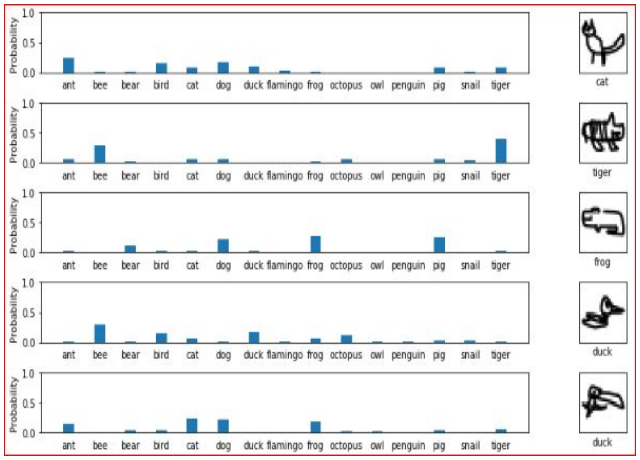


Figure 10: Some examples of predictions



VI. Software Package Description

All codes can be found in our GitHub repository https://github.com/Sapphirine/NiceDrawing. There are three major notebooks under the app folder:

- classification - Built a classification engine for the doodle drawings, as described in earlier sections.
- cartoonify - Generated a doodle drawing from a photograph of animals by utilizing the TensorFlow object detection API. An example is shown in *Fig 11*.
- tsne - Generated t-SNE maps of drawings for each category.

We also created interaction interfaces with GitHub Pages https://irislxxy.github.io/nicedrawing/ and Observable https://beta.observablehq.com/d/5539458d0624933c, which takes an input of a photograph of animals uploaded by the

users and retrieves corresponding t-SNE maps of the categories recognized. An example is shown in *Fig 12*. The brightness level represents the number of drawings in the cluster, i.e., the brighter it is, more drawings there are, and vice versa. And users can explore the drawings for a particular category as shown in Fig 13, and they are also provided with the option to select and download their favorite drawings. A demonstration video can be found at https://youtu.be/rH1-13GFMKM.

Figure 11: Cartoonify photograph of animals

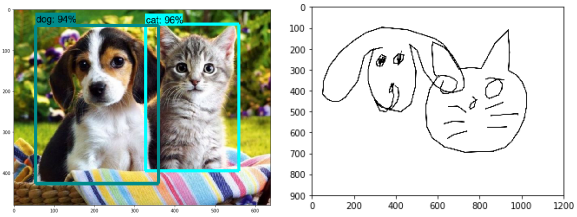


Figure 12: t-SNE of cat (left) and dog (right) datasets

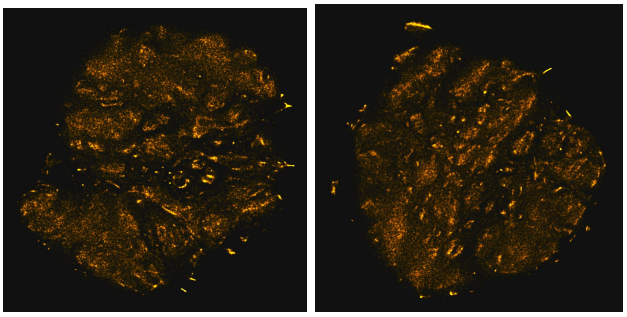
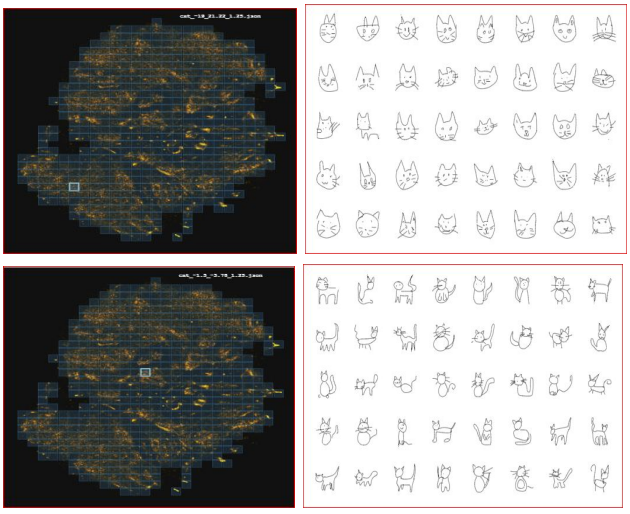


Figure 13:



VII. Conclusion

In this project, we built a classification engine for doodle drawings of 15 animal categories and constructed a web-based application that recognizes the input of a sketched drawing or a photograph and returns an output of thousands similar looking line drawings from the "Quick, Draw!" dataset [3]. The prediction accuracy of our final model was 81.69%, and it could be further enhanced by adding more layers [22, 23]. Also, since the dataset contains 345 categories, we could further expand the scope beyond the 15 animal categories chosen for this project. We could also work on detect and localize not just one but multiple drawing objects in an image, just as the object detection model applied to the real world objects. All team members have contributed equally to this project, and a contribution statement which describes the specific tasks done by each team member can be found in the Appendix.

Appendix

Contribution Statement

- Xueyao Li: Implemented the classification model of the doodle drawings, set up the big data development environment, built the interaction interface with GitHub Pages, organized the GitHub repository.
- Yiyi Zhang: Wrote the final project report, explored possible methods of model enhancement for doodle classification, implemented the cartoonification model and data visualization, built the interactive interface with Observable notebook and recorded the demos.

References

[1] Quick, Draw!. https://quickdraw.withgoogle.com/

[2] AutoDraw. https://experiments.withgoogle.com/autodraw

[3] The Quick, Draw! Dataset. https://github.com/googlecreativelab/quickdraw-dataset

[4] Who is the best at X?. http://rodrigob.github.io/are_we_there_yet/build/

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278-2324, 1998.

[6] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of Neural Network using DropConnect. International Conference on Machine Learning 2013.

[7] D. Ha and D. Eck. A Neural Representation of Sketch Drawings. arXiv: 1704.03477, 2017.

[8] D. Ha, J. Jongenjan, and I. Johnson. Draw Together with a Neural Network. Magenta, 2017.

[9] TensofFlow RNN for Drawing Classification API. https://www.tensorflow.org/tutorials/sequences/recurrent_quickdraw

[10] Kaggle Competition: Quick, Draw! Doodle Recognition Challenge.. https://www.kaggle.com/c/quickdraw-doodle-recognition

[11] O. Russakovsky, J. Deng, H. and etc. ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015.

[12] TensofFlow Object Detection API. https://github.com/tensorflow/models/tree/master/research/object_detection

[13] TensorFlow Image Classification API. https://github.com/tensorflow/tfjs-models/tree/master/mobilenet

[14] L.J.P. van der Maaten and G.E. Hinton. Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research 9(Nov):2579-2605, 2008.

[15] Multicore t-SNE. https://github.com/DmitryUlyanov/Multicore-TSNE

[16] Apache Spark. https://spark.apache.org/

[17] Keras. https://keras.io/

[18] J. Brownlee. Image Augmentation for Deep Learning with Keras, 2016.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15: 1929-1958, 2014.

[20] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv: 1502.03167, 2015.

[21] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv: 1409.1556, 2015.

[22] J. Hestness, S. Narang, N. Ardalani, and etc. Deep Learning Scaling is Predictable, Empirically. arXiv: 1712.00409, 2017.