Quick Draw! Doodle Recognition Deep Learning Strategy

Rui Ning, Yumeng Yue and Zewen Zhang
Department of Materials Science and Engineering
Stanford University
ruining@stanford.edu, yuey3@stanford.edu, zwzhang@stanford.edu

Abstract

Convolutional Neural Networks(CNN) have been shown to be powerful in computer vision tasks like object detection, recognition. In this project, we have explored different neural network architectures to implement "Quick Draw!" game. Three different CNN models are adopted here: 4-layer CNN, MobileNetV2 and DenseNet169. We present qualitative result analysis of these models and conclude that DenseNet169 model with RGB images as samples performs better in classification of drawings with simple features.

1 Introduction

Google released an experimental human-AI interactive game in 2016 titled Quick, Draw!, in which players draw a given object under 20 seconds, and while the user is drawing, an underlying algorithm attempts to guess the category of the object, like “bee”, “backpack”, “ant”, etc. and the predictions evolve as the user adds more and more detail. [2]

This project is aimed to build a extraordinary classifier for the Quick, Draw!’s dataset. Our goal is to build a efficient neural network classifier to recognize over 300 categories from 50M doodles with higher accuracy. In order to achieve high prediction accuracy, smaller size, compatible with mobile devices, and large dataset processing ability, we need a neural network with a high accuracy as well as relatively a simple architecture. And beyond the scope of this task, the exploration of efficient neural network dealing with this task will have an immediate impact on handwriting recognition and its robust applications in areas including OCR (Optical Character Recognition), ASR (Automatic Speech Recognition), and NLP (Natural Language Processing).

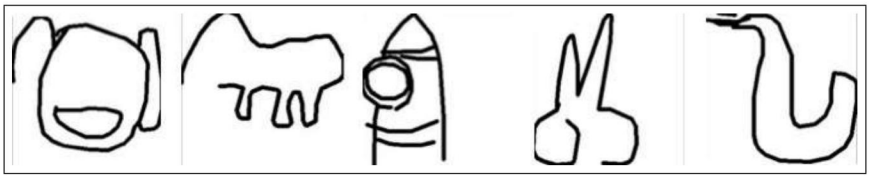

Figure 1: Doodle Image

2 Related Work

For image recognition, Seddati et al. improved sketch recognition using CNN with residuals addition to achieve a mean average accuracy of 79.18% with respect to TU-Berlin sketch benchmark and 93.02 % with respect to the sketchy database, while the human Mean Average Precision was only 73% on the TU-Berlin sketch benchmark.[8]

CS230: Deep Learning, Winter 2019, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

He et al. proposed the idea of residual networks, ResNet, by reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions Deep Residual Learning. They found it easier to train deep convoluted neural networks.[4]

Despite the strong representative power of deep convoluted neural networks, it's generally getting harder and harder to train deeper networks.

Huang and Liu et al. brought about DenseNets architecture where the model adopt the idea of residuals in the network, but connects each layer to every other layer in a feed-forward fashion. They maintained or even exceeded the representative power with a smaller number of parameters as compared with ResNets on benchmark tasks like CIFAR-10 and ImageNet. [6]

Howard et al. presented a class of models called MobileNet for mobile or embedded vision applications. MobileNets are built on a streamlined architecture that use depth-wise separable convolutions to build light weight deep neural networks. Two simple global hyper-parameters are introduced for the trade-off between latency and accuracy with high efficiency. They performed extensive experiments on mobileNet and compared to other models on ImgaeNet classification tasks. Their result demonstrates the effectiveness of MobileNet in a wide range of applications such as object detection, object classification, face attributes, etc. [5]

Based on MobileNet framework, Sandler along with colleagues at Google Inc designed a even more powerful network MobileNetV2 with inverted residual structure where the shortcut connections are between the thin bottleneck layers to remove non-linearities in the narrow layers in order to maintain representational power. [7]

3 Dataset and Features

3.1 Dataset Preparation

The dataset contains 50M images in 300+ categories form the Quick, Draw! players doodles. The drawings were labeled with metadata as what the players were asked to draw and the country they were located. The dataset has two different versions. One is the raw data directly recorded from the user inputs. Another dataset is the simplified version which use less point to define a line. In this project, we ran the neural network models first through the simplified data to modify the parameters, and then use the raw data to further develop the network model to greater accuracy. And here to simplify our model, we didn't take into account the nationality of each doodle's drawer as a variable in prediction as it might not contain as much information as drawing itself.

3.2 Data Processing and Sampling

There are 340 classes in this dataset, and for each class, due to the limitation of the memory of our instance, we varied the samples per class based on how we processed the data in order to fit in the memory. The general way to clean data is to read the points from the input and draw lines between points on an empty image and convert the image into a numpy array. The sample will be labeled and the test set will be split from the input data after preprocessing.

To test our models, we split the data into three different folds: 80% for training, and 20% for testing. To reduce computation time and storage of the data, we decided to create a smaller subset of the original dataset by randomly sampling 2% of the drawings from each category. As a result, we obtain approximately 544,000 examples for the training set and 136,000 examples for he testing set. Furthermore, the number of drawings in each category is balanced, so this leaves approximately 1600 examples per category in the training set.

Apart from raw image, we preprocessed our images to include more information. Simply drawing a greyscale picture would only include information of shape. To better utilize the information encoded in the dataset, we decide to include time stamp of the drawings. Intuitively, people draw a specific pattern in a fixed sequence, which could help the algorithm to recognize image class. Also, in order to fit current convolutional neural network models, which always takes three-channel image as the input, we divide the sequence information into two parts, namely the stroke sequence and the point sequence in each line. Therefore, we construct the three layers of as-generated pseudo-RGB images respectively, the first channel contains shape information, the second channel contains stroke sequence information, and the last channel contains point sequence in each stroke. In Figure3, we

demonstrated the generation of a 3-channel image of a clock from its 1-channel image. The first layer is the original greyscale image, while the second layer is a stroke sequence.

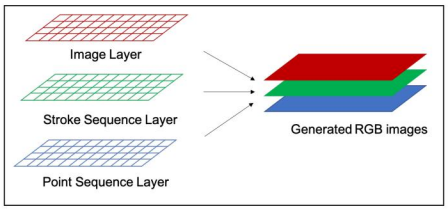

Figure 2: Schematic of data processing

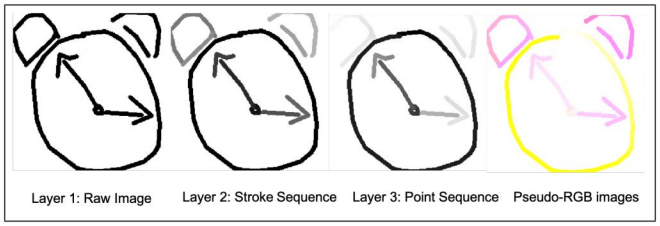

Figure 3: Pseudo RGB Picture Generation

We successfully generated pseudo-RGB pictures as shown in Figure 4. The colored image has a different colors for different strokes, and color variation in each stroke. However, limited by our computation resource, we adopt a resolution of 64*64*1 for greyscale image, and 32*32*3 for RGB image to fit in the memory.

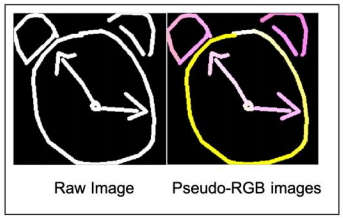

Figure 4: Comparison of raw image and generated image

4 Methods

As a baseline, we worked with a simple 2-layer CNN model. Intuitively, images here are not complex pictures taken from real world, therefore we started off with simple shallow CNN models. The 2 layer CNN started to overfit training set with a test set accuracy of around 60%.

In this 4-layer CNN model, for each layer, we applied a 3*3 filter with same padding followed by a 2*2 max pooling and 0.1 dropout. These 4 layers are followed by 2 fully connected layers. The RELU activation function is applied in each layer and a soft-max prediction function is used for multi-class classification. The architecture is as shown in Figure 5.

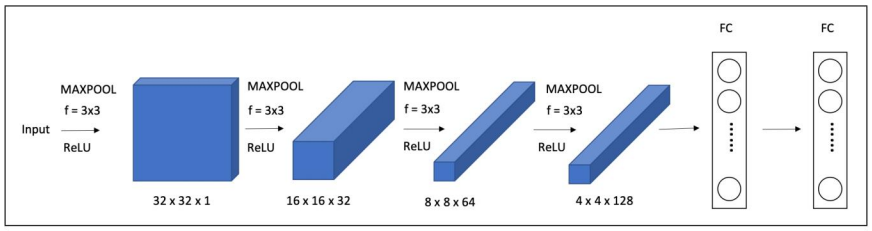

Figure 5: 4-Layer CNN Architecture

After working with as-established 4layer CNN model, we tried out more complex neural network to test out if deeper neural networks could help with prediction accuracy. We have experimented with VGG16, VGG19[9] and ResNets[4]. It turned out that if we laod pretrained parameters with benchmark tasks[1], the model yields random guess results at around 0.003, which indicated that picture features are quite different from real world image. Also, it's too hard to train deep CNNs like ResNet and VGG architectures from scratch.

Therefore, we turned to more well-established and efficient models. We directly adopted the DenseNet169[6] and MobileNetV2[7] architecture from Keras[3] using max pooling for all of the pooling layers. The architecture of the two models are shown in Figure 6 and Figure 7. It's important to note that we did not load any pre-trained weights from other benchmark datasets like ImageNet because those were trained with images that are much more complex than our samples. Therefore, we chose to train our model from scratch. We used mini-batch gradient descend method with batch size of 128 for all the models.

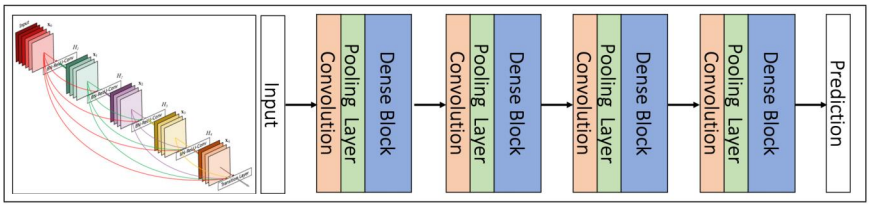

Figure 6: DenseNet Architecture

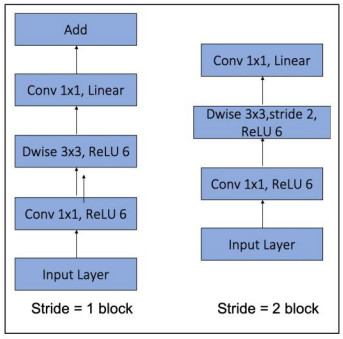

Figure 7: MobileNetV2 Architecture

5 Experiments, Results and Discussion

From Figure 8, we noticed that all models yield an accuracy of higher than 70% despite different types of image samples, respectively. The model with best performance is DenseNet169 trained on pseudo-RGB images.

Surprisingly, the shallow 4layer CNN model worked quite well as compared with more advanced neural networks. While the other two models over-fit quickly only after 5 epochs, MobileNetV2 gradually increase prediction accuracy upon epochs. This probably suggests that we don't need a complex model to make good predictions.

From Table 1, we can conclude that the performance of 4layer CNN model depends less on whether the image is grayscale or RGB compared with DenseNet169 and MobileNetV2 models. With more information contained in the image, the DenseNet169 model performs better while the performance of MobileNetV2 is worse. We believe this is due to the complexity of the model. MobileNetV2 is a light-weight model so when predicting images with less features, it performs better. This also explains that MobileNetV2 did not overfit after 20 epochs while other models overfit quickly. Thus, adding more information into the image will hurt the performance of MobileNetV2 model. On the contrary, DenseNet169, which has a similar representative power as ResNets, performed better when samples are more complicated. Overall, the method of converting samples into pesudo-GRB would be helpful for more complicated and representative models rather than simple and shallow models.

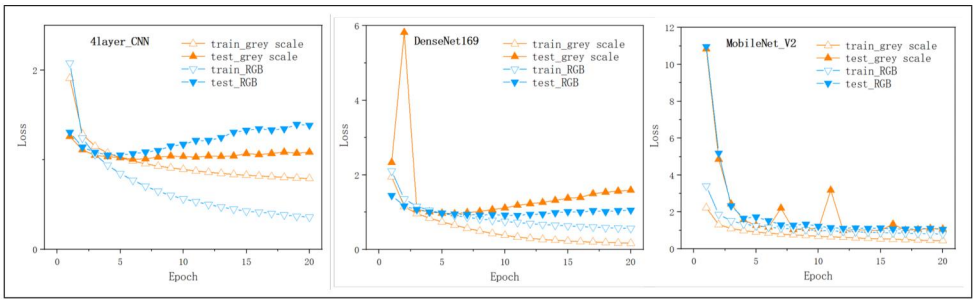

Figure 8: Training and Test Loss

However, it's reasonable to speculate that we need to include more information from the data set or figure out a way to more reasonably include time sequence information.

Accuracy	4layer CNN	DenseNet169	MobileNetV2
1channel-greyscale image	71.8%	71.4%	74.1%
3channel-pseudo RGB image	71.4%	75.0%	70.8%

Table 1: Test Accuracy

We believe that there are several possible reasons for the low test set accuracy: 1. Limited training sample. Here we only use a maximum of 2000 samples of each class to train our model to save time. 2. Large number of classes vs. shallow neural networks. We have over 300 classes, whereas less than five layers of CNN is tested here; 3. Lack of hyperparameter tuning. We didn't carefully tune the hyperparameters, like node numbers of each layer, activation functions, dropout values, etc.; 4. Noisy data. When picking samples from the dataset, we did not filter out some noisy samples which are labeled as "not recognized".

6 Conclusion and Future Work

In conclusion, we have experimented with three different kinds of neural network models, 4layer CNN, MobileNet V2, and DenseNet169, on grayscale and RGB images. In a prediction out of 340 classes, all three models yield over 70% prediction accuracy. The results are fairly good considered that the number of class is very large, yet there are still problems remained and space to improve.

To further improve our model, the most straight-forward way is to train models on an AWS instance with larger RAM, so more samples per-class with higher resolution can be included. We only used around 2% of the total dataset for training, leaving the rest of the dataset open for further development. Also, the dataset we used is the simplified version that some of the information were discarded from the raw version dataset.

We'll also look into test set prediction results and try to have in-depth understanding of the wrong predictions from our models, and make modifications accordingly. To improve our model accuracy, we might work on "1 vs 1" vote strategy or ensemble to increase our prediction accuracy. Lastly, we will try to strike a balance between prediction accuracy and the size of our model, since ultimately we would like to build an efficient classifier that would work on mobile devices with high accuracy.

7 Contribution

We processed image data, implemented the training process and evaluated models. All authors contributed equally on these works. The authors thank Ahmadreza Momeni for his constructive suggestions, careful guidance and dedicated mentorship.

8 Code Link

https://github.com/zwzhang2018/cs230-final-project

References

[1] Imagenet. `http://www.image-net.org/`. Accessed Feb 01, 2019.

[2] Quick draw. `https://www.kaggle.com/c/quickdraw-doodle-recognition`. Accessed Feb 01, 2019.

[3] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[4] 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.

[5] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[6] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[7] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[8] Omar Seddati, Stéphane Dupont, and Saïd Mahmoudi. Deepsketch 3. *Multimedia Tools Appl.*, 76(21):22333–22359, November 2017.

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