QuickDraw Doodle Recognition System

Course: CS 256, Topics in Artificial Intelligence

Guidance: Dr. Gayathri Namasivayam

Project By
Aparna Kale
Rajeshwari Chandratre

***Abstract*—This Doodle Recognition System aims to recognize the ten different categories of Doodles, which are given as inputs to this system in the form of vectors. The output is in the form of two guesses for every doodle to be identified. We have utilized the Google Quickdraw challenge dataset to build our system.**

Keywords—Doodles, vectors, MobileNet, Depthwise convolution, images

I. Introduction

"Quick, Draw!" was released as an experimental game by Google to demonstrate the working of AI in a playful manner to the commoners. The game prompts users to draw an image depicting a certain category, such as ”house,” “table,” etc. and the system recognizes what the user has drawn. The game generated more than 1B drawings, of which a subset was publicly released. We utilize this dataset made available by Google Creative Lab to build our own ‘Doodle Recognition System.’

We give input as the vectorized form of black and white drawings drawn by users from across the world. This input is fed to the system and the system outputs the doodle category in the form of two best guesses. We used the MobileNet approach, a deep learning approach to recognizing QuickDraw Game Doodles using depthwise convolution neural networks.

II. Data Exploration

A. Selecting the dataset

The Quick Draw Dataset is a collection of 50 million drawings across 345 categories, contributed by players of the game Quick,Draw! The drawings were captured as timestamped vectors, tagged with metadata including what the player was asked to draw and in which country the player was located.

We are utilizing the dataset used for the Google ‘Quick, Draw!’ Doodle Recognition Challenge [1]. However, we are limiting our scope to only ten categories of doodles. The available raw dataset is in the form of CSV files, where each file consists of different columns, which we will see in the next section.

B. Challenges in the dataset

Since the training data comes from the ‘Quick! Draw’ dataset itself, drawings in the dataset can be incomplete or may not match the label. Hence, there is a need to build a recognizer that can effectively learn from this noisy data and perform well.

Another challenge was that the test data we was unsupervised, meaning that the drawing records were not labelled as per their categories. Since there was a certain difficulty level associated with the process of unsupervised dataset recognition and then to compute of accuracy level of this unsupervised test data, we decided to use some fraction of available training data as our training dataset. We will see the details associated with this split further in Section IV.

C. Structure of the Data

Each CSV file contains data belonging to only one type of drawing. The raw data set is available in the following format:

Key	Type	Description
key_id	64-bit unsigned integer	A unique identifier across all drawings.
word	string	Category the player was prompted to draw.
recognized	boolean	Whether the word was recognized by the game.
timestamp	datetime	When the drawing was created.
countrycode	string	A two-letter country code (ISO 3166-1 alpha-2) of where the player was located.
drawing	string	A JSON array representing the vector drawing

Fig. 1. Dataset Structure

III. Scope of the Project

A. Categories of the drawings considered

In our project, because of the time and complexity constraints, we consider 10 drawing categories. They are as follows:

a. Airplane
b. Book
c. Cake
d. Candle
e. Fan
f. Fork
g. Hand
h. Hat
i. House
j. Lollipop

Thus, we train the model with the dataset containing drawings from the mentioned 10 categories and test the data with the drawings that belong to these classes.

B. Available Data

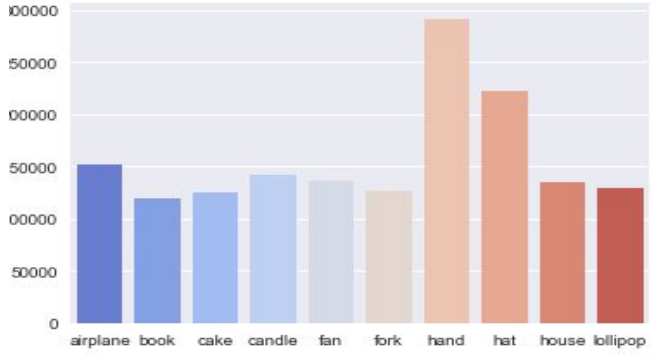

Fig. 2. Data Available for Training

Fig.2 was plotted as a part of our data exploration process. The plot shows the number of records available vs. the categories of Doodles. Here, we understand that the dataset contains the unequal distribution of data over the categories of Doodles meaning that we do not have the equal number of records in each category to train our model. This data is as per the dataset we got from the 'QuickDraw' dataset.

IV. Data Preprocessing

A. Shuffling the dataset

We have our entire data in the form of CSV files and each CSV file has the records belonging to a specific category of Doodle. Once we got the 10 CSV files containing 10 doodles of the categories that we are considering, the most important part of data preprocessing was to merge all the CSVs in such a way that the drawing vector should get evenly distributed and easier to feed to the model. Example training sample from airplane.csv:

A	B	C	D	E	F
countrycode	drawing	key_id	recognized	timestamp	word
US	[[[167, 109, 8	5.1528E+15	TRUE	12:07.3	airplane
CA	[[[2, 14, 34, 1	6.6229E+15	FALSE	39:04.7	airplane
US	[[[90, 88, 95,	6.577E+15	TRUE	08:35.2	airplane
US	[[[82, 49, 15,	5.6432E+15	TRUE	35:17.5	airplane
IL	[[[64, 38, 23,	6.67E+15	TRUE	11:11.7	airplane

Fig. 2 Before shuffling data, columns in the airplane.csv

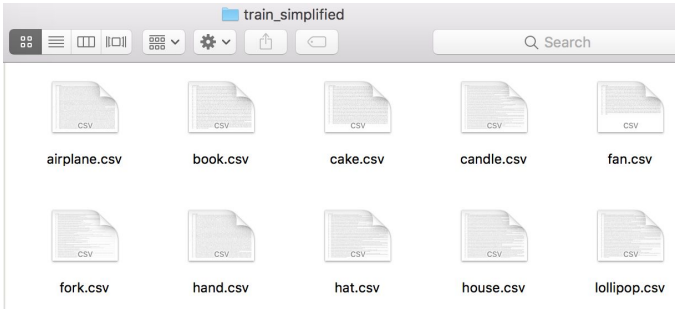

Fig. 3 Raw Data available before preprocessing

We made sure to pick the data from each of the categories and put in one chunk.csv.gz file and also the content of this file was shuffled making sure that all the categories will be the part of each chunk.csv.gz

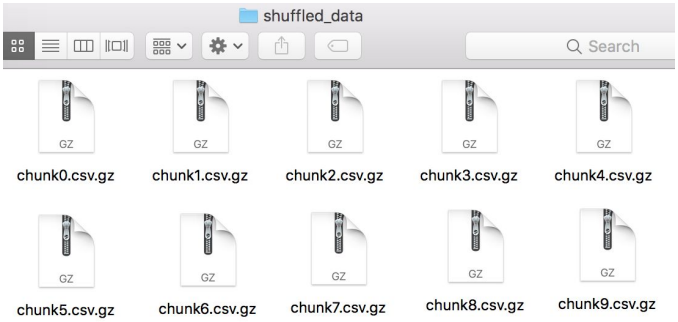

Fig. 4 Data after shuffling the records

The example training sample from chunk.csv:

A	B	C	D	E	F
countrycode	drawing	recognized	timestamp	word	y
US	[[[61, 57, 49,	TRUE	25:26.1	hand	6
US	[[[55, 55], [0	FALSE	52:52.6	fan	4
US	[[[187, 121, 1	TRUE	53:25.8	house	8
CZ	[[[132, 86, 6(	TRUE	22:26.5	fan	4

Fig. 5 After shuffling data, columns in the chunk.csv

B. Feature Selection

In shuffled .CSV, We also have temp column which we add while processing but that is removed later. We dropped key_id column in the chunking step itself. Column

'y' (Fig. 5) representing the index of the category of the drawing record is added for every record. This will hold the index of the category the drawing belongs to. For example, if the drawing is of the airplane, the 'y' field will take a value of '0', and so on. Further, we decided to pick only the relevant features necessary for the model for prediction. In our case 'drawing' and newly added 'y'. We drop the irrelevant columns such as 'countrycode', 'timestamp', 'recognized'. These columns do not add any significance to our data that is required for the model to predict the data.

C. Splitting the DataSet

As previously mentioned, we had an unsupervised testing data. Hence we decided to split a fraction of our training set into the test data set. We split this now available clean and supervised training data into train, test, and validation sets. Thus we now have eight CSVs for training the data, one csv for validation set, and one CSV as a testing set.
Following is the distribution of dataset:
Total number of doodles = 1,578,324
Total number of training samples = 1,262,745
Total number of validation samples = 157,590
Total number of test samples = 157,989

V. Model & Model selection

A. Model Selection & Literature Survey

Model selection is a crucial part of any machine learning project. We faced a couple of challenges in selecting the best model. There are a couple of approaches here when it comes to image processing. First, is using the existing approach such as MultiLayer Perceptron, AlexNet using 9 layers. Literature survey suggested that the accuracy as low as 0.4 and 0.7 for the specific use case. Second, is transfer learning. Keras provides pre-trained models including GG19, ResNet50, Inception v3, Xception, MobileNet. However, MobileNet is essentially a streamlined version of the Xception architecture optimized for mobile applications. In the case of transfer learning, one has to modify the last layer of the existing model. The third approach is to train the model with weights as "none". We then came across Google's MobileNet paper[2] which suggested why MobileNet is efficient. The paper also mentions the hyperparameters to be set for the model. After understanding it's reference, we decided to use the hyperparameters mentioned in the paper and trained our model with no weights.

Initially, we trained the data on the complete train file and decided to rebuild the test data with the labels, However, it got difficult to rebuild each and every doodle. So we discarded this model. After that, we split the actual data in the three chunks of 12L, 1.5L, and 1.5L of data for train, validation and test respectively as mentioned previously in Section IV. We then saved the model and weights using model.save().

B. Why MobileNet?

Convolutional neural networks became popular in computer vision ever since AlexNet's deep CNN. The general trend to achieve the accuracy is to add the convolutional layers but it does not make it. A different approach for obtaining small networks is shrinking, factorizing or compressing pretrained networks. Compression based on product quantization, hashing, and pruning, vector quantization, and Huffman coding has been proposed in the literature.[2]

The MobileNet model is based on depthwise separable convolutions which is a form of factorized convolutions[2]. MobileNets are based on a streamlined architecture that uses depth wise separable convolutions to build lightweight deep neural networks. Its proven to be the across a wide range of applications and use cases.The idea behind MobileNet is that convolutional layers, which are essential to computer vision tasks but are quite expensive to compute, can be replaced by depthwise separable convolutions. The convolution layer is split into two parts(Fig. 6,7). One is the depthwise convolution layer which filters input followed by a pointwise convolution layer that combines these filtered values to create new features. Pointwise convolution, a simple 1×1 convolution, is then used to create a linear combination of the output of the depthwise layer. MobileNets use both batchnorm and ReLU nonlinearities for both layers[2]. When added together, it forms a "depthwise separable" convolution block. It does approximately the same thing as traditional convolution but is much faster.

$$\text{The complexity of Depthwise Convolution} = Dk.Dk.M.Df.Df + M.N.Df.Df$$

$$\text{The complexity of Standard Convolution} = \text{Cost} = Dk.Dk.M.N.Df.Df$$

where Dk is the filter of size k, input channels = M, the number of output channels = N, the feature map = DF × DF [2]

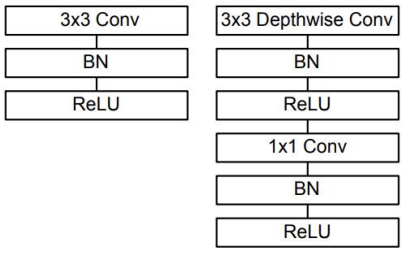

Fig.6 Convolution NN vs Depthwise NN layer split

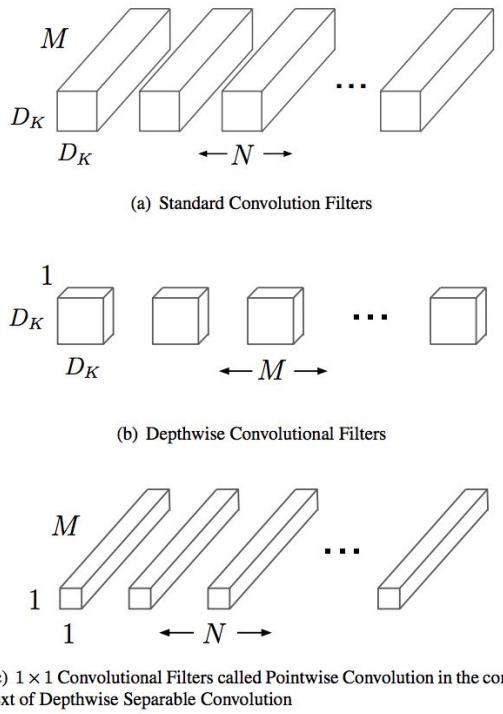

Fig. 7 Comparison between a Simple Convolution Layer and MobileNet's Depthwise Convolution Layer [2]

C. Model Parameters

MobileNet architecture uses two hyper-parameters width multiplier and resolution multiplier. Width multiplier has the effect of reducing computational cost. The role of the width multiplier α is to thin a network uniformly at each layer. For a given layer and width multiplier α, the number of input channels M becomes αM and the number of output channels N becomes αN[2]. The paper [2] demonstrates the performance of MobileNets using `alpha` values of 1.0 (also called 100 % MobileNet). So, we used alpha = 1.0 as our hyper parameter which is also called as the width multiplier. Original MobileNet is trained on 1000 classes. Since we are classifying doodle in 10 categories, we set `classes` = 10. Other parameters used for the the model and fit_generator()[3] are as follows:

1. save_best_only = True - We wanted to save model weights and model itself for future use.
2. steps_per_epoch = 800
3. epochs = 3
4. Loss = categorical_crossentropy
5. Optimizer Used = Adam with LR 0.002
6. max mode = learning rate will be reduced when the quantity monitored has stopped increasing.
7. input_shape = resized to 64X64 from 256X256

For callback, we used the function called ReduceLROnPlateau and monitored validation categorical accuracy. This helps in reducing the learning rate when the metric stops improving.

VI. Design

The image Fig. 8 shows the design or the workflow of our system, where we gathered the dataset in the form of CSV files, where each CSV file contains data related to a specific different category which is shuffles into 10 different chunks. Each chunk holds the records belonging to all the 10 categories distributed randomly. Each drawing vector present in the vector is a combination of strokes.

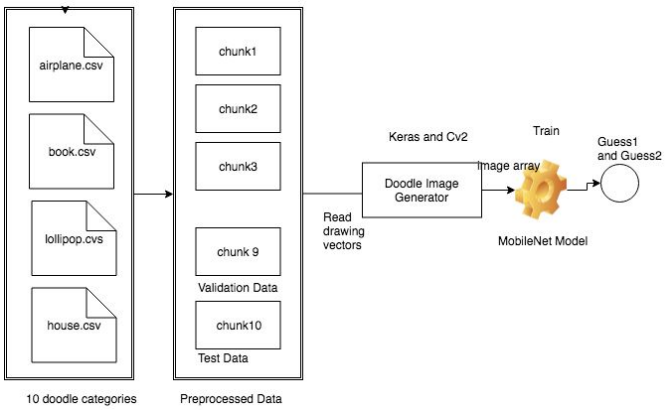

Fig. 8 Workflow of our System

First, we generated a grid of size 64X64 filled with zeros. We fed every stroke of the drawing to cv2.line() to generate the lines by joining the coordinates present in the given stroke. We did this for every stroke of the drawing and update the image. When a line is drawn from A to B, it contains infinite points. Instead, the simplified line is given as an input which will only have two coordinates i.e. A and B. This makes the image lightweight. We fed this array to the preprocess_input()[3] function provided by Keras for MobileNet. In order to train the model, we fed the preprocessed_input to the MobileNet model. After the MobileNet processes the input records, it provides the output in terms of 2 guesses for every image record. This was made possible by the Keras.metrics which allowed us to set k=2 for top_k_accuracy. We considered that our prediction is correct if one of the either or both the guesses is correct.

VII. Results And Accuracy

A. Training Accuracy

For doodle recognition, we achieved maximum accuracy. While building the model in the initial phase we tried and tested different parameters for the model and we set epochs=15. With 15 epochs we achieved, accuracy as 0.99 after training the model for 24 hours. We then reduced our epoch size to 3 and could achieve accuracy as high as 0.97. Following matrix shows the accuracies and losses at the time of training the model

Validation Set: categorical_crossentropy	0.1985
Validation Set: categorical_accuracy	0.9456
Validation Set: top_two_accuracy	0.9767
Training Set: top_two_accuracy:	0.9797
Training Set: categorical_accuracy	0.9516
Training Set: categorical_crossentropy	0.1582

We trained the model using fit_generator() [3] method used in keras. It takes the data batch by batch. Moreover, it returns the history object that records training loss values and metric values at successive epochs along with validation loss and validation metrics. Using this history object we plotted the graphs for the epochs.

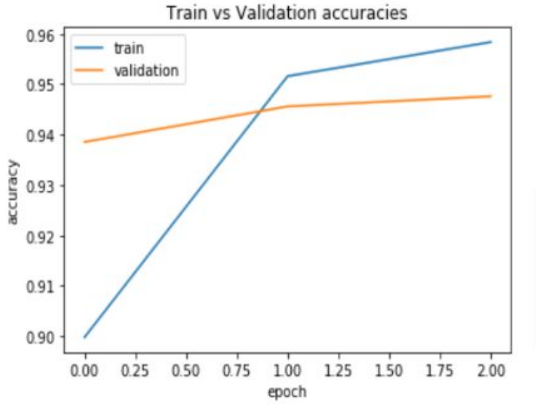

Fig. 9 Train vs. Validation Loss plotted against epoch size

As shown in Fig. 9, our training accuracy increases as with every new epoch. It increases from 0.9 to 0.96 as the epoch increases from 0 to 2. Similarly, there also is a slight increase in the validation accuracy with the growing epoch.

As it can be observed in Fig.10, the training error decreases as the epoch size increases. Moreover, the validation loss also seems to decrease with every growing epoch.

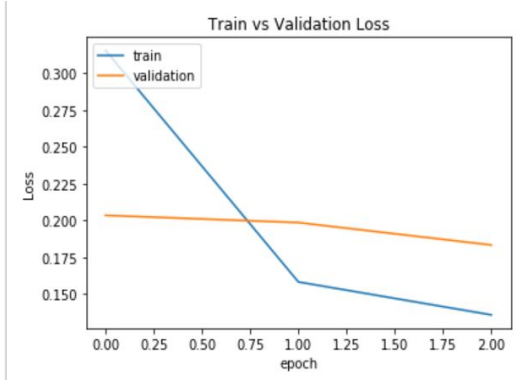

Fig. 10 Train vs. Validation Loss plotted against the number of epochs

B. Testing Accuracy

In order to get the total accuracy, we added the accuracies of both of the guesses, i.e. the guess 1 accuracy and guess 2 accuracy, and consider this as the total test accuracy score.
Testing Accuracies:

Doodle Guess 1: Doodle Guess 2: Total Test Accuracy	0.9469330143237821 0.029464076612928748 0.9763970909367108

VIII. Future Scope

We limited the scope of our model to 10 categories in order to keep the reduce the complexity of the model. We can consider adding more categories of the doodles. Also, with the current dataset we can also consider the features such as the a number of strokes into the account while building the complex model.
We can also enhance the working of this model by making it more interactive in terms of identifying the doodle immediately when someone draws it on the screen.

Hence we can increase the future course of our project by doing the following::

1. Add more categories of the images to be recognized.
2. Retain features such as strokes to find complex doodles such as flower vs blackberry
3. Simulating the quickdraw game and guess the doodle continuously for n guesses.
4. Draw a doodle and capture it using WebCam to identify the doodle.

IX. References

[1] Dataset Reference:
https://github.com/googlecreativelab/quickdraw-dataset
[2] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Applications. Google Inc. arXiv:1704.04861v1.
https://arxiv.org/abs/1704.04861
[3]https://keras.io/applications/#mobilenet

X. Acknowledgment

We sincerely thank Dr. Namasivayam for her invaluable guidance to us and helping us in various aspects such as selecting the topic, to catering to our doubts regarding the difficulties faced in the process.