COMP 576 Final Report
Classification of Quick, Draw! dataset

Yuhui Tong (yt30), Chen Zeng (cz39), Jianwei Jin (jj56)

December 9, 2018

1 Introduction

1.1 Background Introduction

This report covers the classification of Quick Draw dataset [1], a collection of 50 million drawings across 345 categories, contributed by players of the game Quick, Draw! The drawings collected in Quick, Draw! game were captured as timestamped vectors, tagged with metadata including what the player was asked to draw and in which country the player was located. Figure 1 shows some drawing examples. The task of this project is to build a better classifier for the existing Quick, Draw!

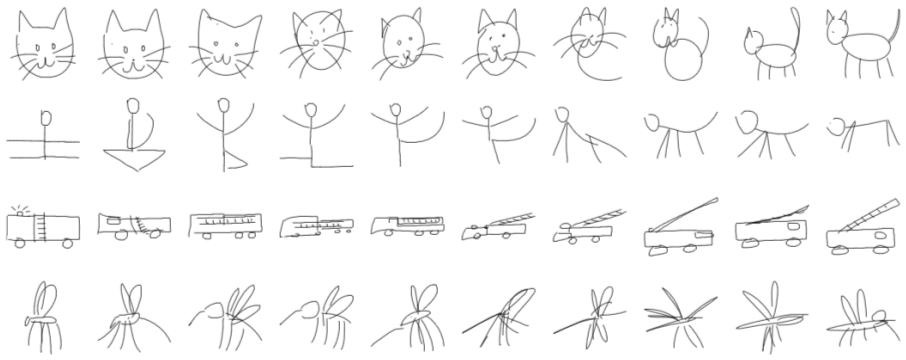

Figure 1: Examples of Quick, Draw! data set, for cat, person, fire truck and mosquito.

dataset. Considering the uniqueness of this dataset (detailed introduced in Sec. 1.2), the advancing models on this dataset can improve pattern recognition solutions in a broad way. This will have an immediate impact on handwriting recognition and its robust applications in areas including OCR (Optical Character Recognition), ASR (Automatic Speech Recognition) & NLP (Natural Language Processing) [2].

1.2 Dataset

The Quick, Draw! dataset has a very interesting way of data representation.

1.2.1 Data format

The raw data of Quick, Draw! dataset is available as ndjson files seperated by category as Figure 2 [3].

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

Figure 2: Key, Type, and Description of the Quick, Draw! dataset.

Each sample contains one drawing. An example of a single data sample is showed in Figure 3.


```
{
  "key_id":"5891796615823360",
  "word":"nose",
  "countrycode":"AE",
  "timestamp":"2017-03-01 20:41:36.70725 UTC",
  "recognized":true,
  "drawing":[[[129,128,129,129,130,130,131,132,132,133,133,133,133,...]]]
}
```


Figure 3: An example of a single data sample.

The format of the drawing array is showed in Figure 4.

```
[
  [  // 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
]
```

Figure 4: The format of the drawing array.

As Figure 2, Figure 3, and Figure 4 show, the doodles are represented by a sequence of strokes. In the simplified dataset, timing information is removed and a stroke is presented by sampling points on this stroke with one pixel spacing. The order of strokes in dataset simply indicates the time order of which strokes are drew first.

1.2.2 Two Types of Dataset: Raw / Simplified

Raw data contains more information than simplified data. The raw drawings can have vastly different bounding boxes and number of points due to the different devices used for display and input while

simplified version has positioned and scaled the data into a 256x256 region. This is the simplification processing to illustrate the differences between raw and simplified data:

- Align the drawing to the top-left corner, to have minimum values of 0.
- Uniformly scale the drawing, to have a maximum value of 255.
- Resample all strokes with a 1 pixel spacing.
- Simplify all strokes using the Ramer-Douglas-Peucker algorithm with an epsilon value of 2.0.

We list in the table below the difference between raw and simplified data in Table 1

	raw	simplified
timestamp	contained	removed
image size	arbitrary	256*256
data size	65.9G	7.4G

Table 1: The difference between raw and simplified data.

1.3 Models

1.3.1 Challenges

Since the training data comes from the game itself, drawings can be incomplete or may not match the label. The major challenge is to build a recognizer that can effectively learn from this noisy data and perform well on a manually-labeled test set from a different distribution.

1.3.2 Stroke Based Model

Since we have time information which also matters in this specific problem, we want to utilize this information in classification process. The drawings in the raw dataset are represented by a sequence of strokes. Every stroke is represented by a list of points on this stroke with a timestamp for each point. However, the order of the strokes contains enough information for time sequence model which is preserved in the simplified data. Thus, we can process the simplified data and index the strokes by the order they were drew and feed them into the model.

1.3.3 Image Based Models

The drawings are represented as strokes in the simplified data. However, image based model like CNN and MobileNet requires a whole image matrix as input. We can generate the image matrix by connecting the points on the same stroke and also between different strokes. Although the generated image matrix will lose the time information, it is not necessary with CV based model.

1.4 Report Organization

While in this section, we briefly introduced the background of classification problem of Quick, Draw! dataset, the rest of the report is organized as follow:
In Sec. 2, we will review the three approaches we are going to use for the classification problem, namely, CNN, MobileNet and RNN. We will then delve into more details on how we build and train our models and report relevant results in Sec. 3, whereas Sec. 4 will be devoted to a summary.

2 Approaches

2.1 CNN

2.1.1 Reason

The nature of this problem is image classification in which we need to assign a class to each input image within a given fixed set of categories.The CNN architecture implicitly combines the benefits obtained by a standard neural network training with the convolution operation to efficiently classify images. Further, being a neural network, the CNN are also scalable for large datasets, which meets the needs of this problem.

2.1.2 Brief Introduction

Convolutional neural networks (CNN) is a special architecture of artificial neural networks, proposed by Yann LeCun in 1988. CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification. Figure 5 shows a typical CNN architecture for MNIST digit recognition problem.

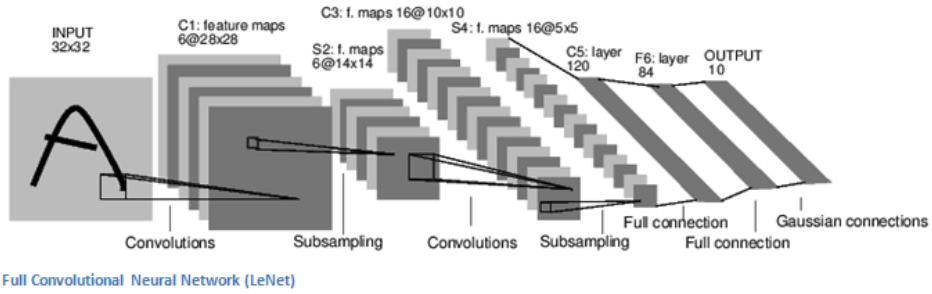

Figure 5: CNN architecture for MNIST digit recognition.

- **Input Layer**: This layer holds the raw input of image with width 32, height 32 and depth.
- **Convolution and Activation Function Layer**: This layer computes the output volume by computing dot product between all filters and image patch, then apply activation function to add nonlinearity. This layer will extract small features from the images.
- **Pool Layer**: This layer is to reduce the size of volume which makes the computation fast reduces memory and also prevents from overfitting.
- **Fully-connected Layer**: This layer will accomplish the high-level reasoning in the neural network. Neurons in a fully connected layer have connections to all activations in the previous layer. Scores of classes will be generated in this layer.

2.1.3 Challenges

Computational Complexity: The size of simplified training data is 30G which contains lots of images. To feed the drawing data into model, we need to convert the strokes into 256*256 image first which also greatly increase the input size. It will be difficult for us to try some deep architectures with limited computational resources and time.

2.2 MobileNet

2.2.1 Reason

Compared to CNN, MobileNet utilizes the depth wise separable convolutions which drastically reducing computation and model size. Based on a streamlined architecture MobileNet uses depth wise separable convolutions to build light weight deep neural networks. It has two global hyperparameters, i.e., width multiplier and resolution multiplier, which can be adjusted to trade off a reasonable amount of accuracy to reduce size and latency.

2.2.2 Brief Introduction

MobileNet is presented by research scientists in the paper *Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications.*. It is a class of efficient models for mobile and embedded vision applications.
Normally, a standard convolution makes both filters and combines inputs into a new set of outputs in one step. In MobileNet, the depthwise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorization has the effect of drastically reducing computation and model size.

2.2.3 Computational Complex Analysis

Suppose a standard convolutional layer takes as input a $D_F \cdot D_F \cdot M$ feature map and produces a $D_F \cdot D_F \cdot N$ feature map. Then the standard convolutions have the computational cost of $D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$, where D_K is the kernel size. Compared with it, the depthwise convolution has a computational cost of $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$ and the combination layer has cost of $M \cdot N \cdot D_F \cdot D_F$. By expressing convolution as a two step process of filtering and combining we get a reduction in computation of $\frac{1}{N} + \frac{1}{D_k^2}$ and the computational cost is obviously reduced.

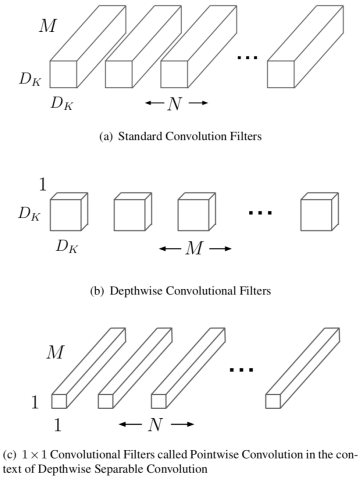

Figure 6: The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

2.3 RNN

2.3.1 Reason

Previous two approaches clearly only makes use of the image information. Intuitively, we want to make use of of the timestamp information as well. Since the drawing are presented by a sequence of strokes, a natural alternative to CNN is RNN, which allows to exhibit temporal dynamic behavior for a time sequence. Hopefully, this approach could provide some extra understanding to the problem.

2.3.2 Brief Introduction

Recurrent Neural Networks or RNN take the previous output or hidden states as inputs. As their intermediate values (state) can store information about past inputs for a time that is not a priori, RNN allows for processing of variable length inputs and outputs. The most popular RNN unit is Long Short-Term Memory (LSTM) units, which overcomes the exploding and vanishing gradient problems that can be encountered when training traditional RNNs. For this reason, we will use LSTM units in our model.

3 Models

Hardware and Software We used Tensorflow and Keras to build our models, which were trained on Google Colab and Kaggle Kernel, both with GPU (Nvidia K80) acceleration.

3.1 CNN

3.1.1 Structure

Figure 7 shows the general computational graph for our basic CNN model which comes from a kernel [4] on kaggle(parameters may not be exact since we tuned some of them). The inputs are 32*32(resized from 256*256) doodle images. Three convolutional layers aims to extract low level features and two fully connected layers construct high level abstraction. We also insert a layer of drop out to prevent overfit. Finally, output the prediction of 340 possible categories.

3.1.2 Model Selection

For convolutional neural network, there are some common hyper parameters we can consider like learning rate, number of convolutional layers, number of filters, types of activation functions, dropout rate. We only choose learning rate, number of filters and dropout rate to tune as below because most of our limited computational resource has been consumed by the huge dataset of this problem.

Learning Rate Figure 8 shows the change of training loss, training and validation accuracies with epoches for different learning rate(we only shows four of them). Graph a, b, c, d show situations when learning rate = e-5, e-4, e-3, e-2 respectively. The top validation accuracies are 0.52, 0.65, 0.68, 0.3 respectively. We found that with the increasing number of epochs, both accuracies keep increasing and training loss keeps decreasing. In the four graphs we presented, the gap between training and validation loss also become smaller gradually. However, we found that the accuracies stop increasing and keep going flat around the 100th epoch(much smaller when learning rate lands on a very bad choose). The top validation accuracy of graph a and d is apparent much lower than the other two(lower than 0.5) which clearly shows that we already hit the edge. We also noticed that with the increasing learning rate, the accuracies and training loss shows a little unstable in the early stage of training process.This may indicates that the learning rates are indeed too large.

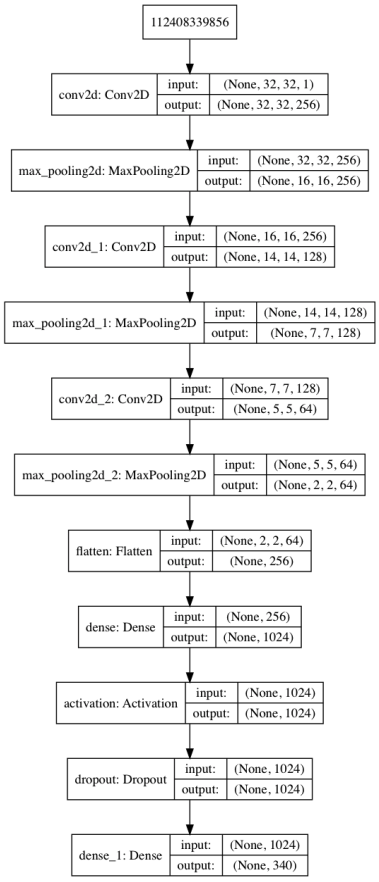

Figure 7: Computational graph for CNN models.

Our results showed that a learning rate of 2.5 * e-3 gave us a smooth curve and highest validation accuracy.

Number of Filters Figure 9 shows the change of training loss, training and validation accuracies with epoches for different number of filters(we only shows two of them). Graph a, b show situations when number of filters in convolutional layers = 128, 256 respectively. The top validation accuracies are 0.7, 0.65 respectively. We found that top validation accuracy of both reached 0.6 at around 70th epoch, but stop increasing after that. Compared with original 64 filters, they require less epoches to train reaching the same validation accuracy as original graph. During our tuning process, we found 128 filters is a best choose, not necessary increase the top validation accuracy but make the model be trained much easier.

Dropout Rate Figure 10 shows the change of training loss, training and validation accuracies with epoches for different dropout rate(we only shows three of them). Graph a, b, c show situations when dropout rate = 0.25, 0.35, 0.5 respectively. The top validation accuracies are 0.7, 0.66, 0.6 respectively. We can see that when dropout get close to 0.5, the gap between training and validation loss become larger and top validation accuracy drops to 0.6. This tells us the dropout rate 0.5 is too large. During our tuning process, we found a dropout rate of 0.25 is a best choose which gives us a almost 0.7 validation accuracy at around 70th epoch.

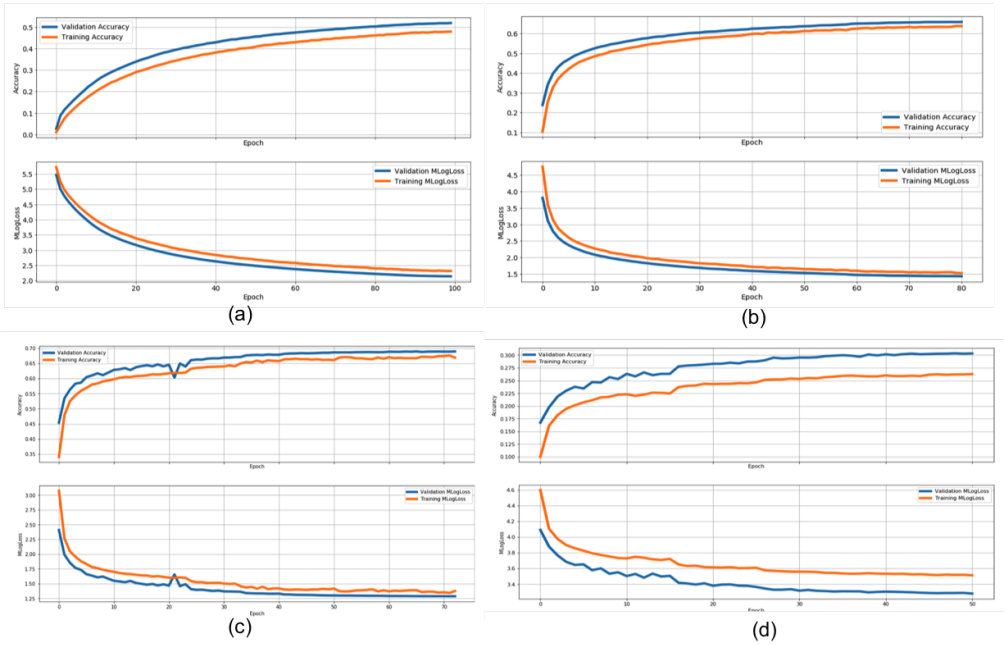

Figure 8: Accuracies and training loss with different learning rate.

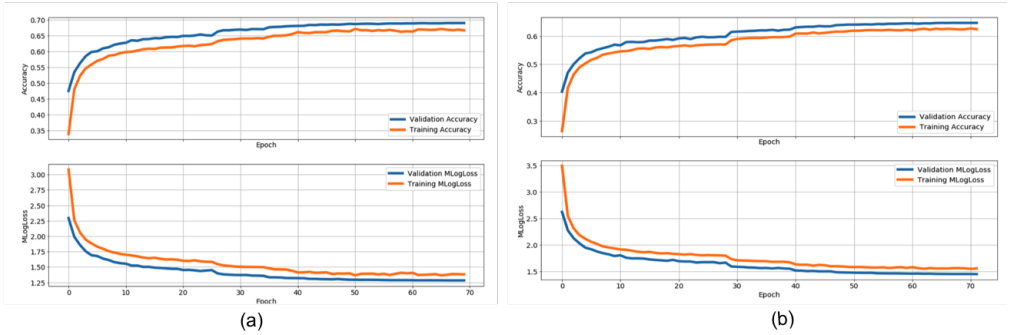

Figure 9: Accuracies and training loss with different number of filters.

3.1.3 Conclusion

Since deep CNN model will require huge computational resource and a amount of time. It's not practical for us to tune these parameters on such a CNN model with limited resource and time. As a result, we set a general CNN model which only has three convolutional layers to tune the parameters. In the whole model selection process, we focus on the learning rate, number of filters and the dropout rate. Our results showed that a combination of learning rate = 2.5*e-3, number of filters = 128, dropout rate = 0.25 can give us a high performance model whose top validation accuracy can reach 0.76.

3.2 MobileNet

3.2.1 Structure

Figure 11 describes the basic graph flow of MobileNet [6]. This model uses depthwise separable convolutions as mentioned in the last subsection. Because of the reducing of parameters, it can be

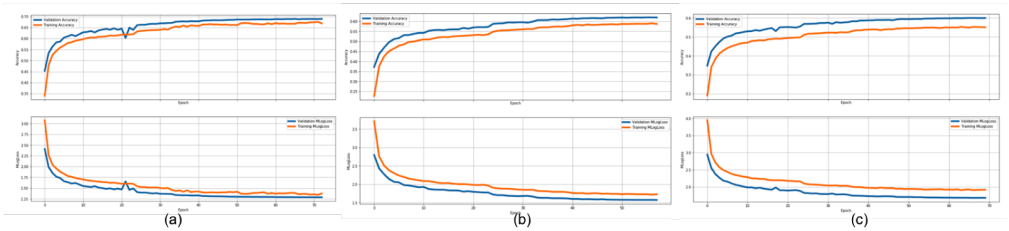

Figure 10: Accuracies and training loss with different dropout rate.

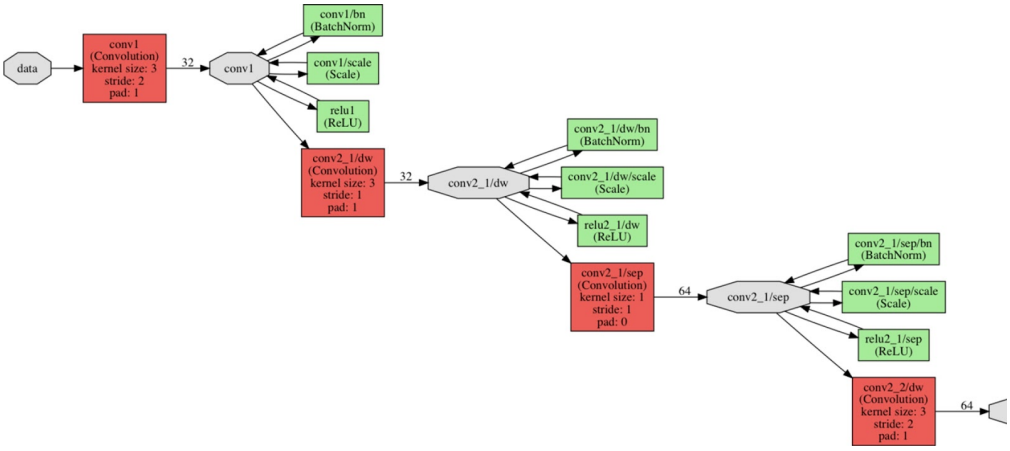

Figure 11: Computational Graph of MobileNet in Doodle Application.

easily run on iPhone or other mobile devices.

3.2.2 Parameters

As mentioned above, the MobileNet can significantly reduce the number of parameters. As for the model in Figure 3.2.1, the number of total params is 3,576,788. And it's consisted with 3,554,900 trainable params and 21,888 non-trainable params.

3.2.3 Model Selection

batchsize We select the batchsize as 680, 800, and 1000, and then record and compare the results of Map3, Accuracy, and Loss. The results of Map3 is recorded in Table 2 and the results of Accuracy and Loss is recorded in Figure 12.

batchsize	680	800	1000
Map3	0.800	0.801	0.807

Table 2: Value of Map3 according to different batchsize.

In Figure 12, (A) describes the Accuracy and Loss change as epochs grow when batchsize = 680, (B) describes the Accuracy and Loss change as epochs grow when batchsize = 800, and (C) describes the Accuracy and Loss change as epochs grow when batchsize = 1000.
From Table 2 and Figure 12, we can find that as batchsize increases, the Map3 and Accuracy will all increase. But this influence of batchsize is not that obvious. Comparing these three pair of

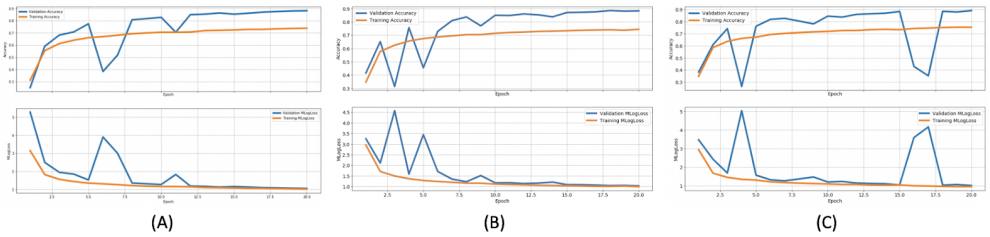

Figure 12: (A) Accuracy and Loss when batchsize = 680, (B) Accuracy and Loss when batchsize = 800, (C) Accuracy and Loss when batchsize = 1000.

parameters, we would like to choose batchsize = 1000 for our final model. Also, as epochs grow, the Accuracy and Loss will be more and more stable.

STEPS We select the STEPS as 600, 700, and 800, and then record and compare the results of Map3, Accuracy, and Loss. The results of Map3 is recorded in Table 3 and the results of Accuracy and Loss is recorded in Figure 13.

STEPS	600	700	800
Map3	0.800	0.796	0.815

Table 3: Value of Map3 according to different STEPS.

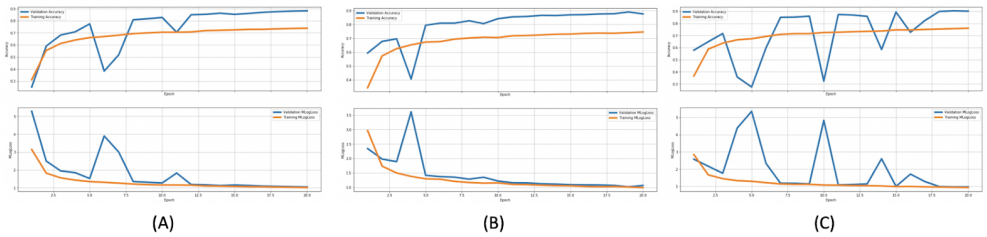

Figure 13: (A) Accuracy and Loss when STEPS = 600, (B) Accuracy and Loss when STEPS = 700, (C) Accuracy and Loss when STEPS = 800.

In Figure 13, (A) describes the Accuracy and Loss change as epochs grow when STEPS = 600, (B) describes the Accuracy and Loss change as epochs grow when STEPS = 700, and (C) describes the Accuracy and Loss change as epochs grow when STEPS = 800.
From Table 2 and Figure 13, we can find that as STEPS increases, the Map3 and Accuracy will all increase. And this influence of STEPS is very obvious. Comparing these three pair of parameters, we would like to choose STEPS = 800 for our final model. Also, as epochs grow, the Accuracy and Loss will be more and more stable.

3.2.4 Results

As for model MobileNet, the results are submitted three times with modification on batchsize and STEPS. The highest accuray on testset is about 0.88 [7]. The submission results are shown in Figure 14.

Submission and Description	Private Score	Public Score	Use for Final Score
gs_mn_submission_8343.csv 25 days ago by Chen Zeng add submission details	0.87996	0.88172	☐
gs_mn_submission_8311.csv a month ago by Chen Zeng add submission details	0.87344	0.87430	☐
gs_mn_submission_7881.csv a month ago by Chen Zeng add submission details	0.82939	0.83099	☐

Figure 14: Accuracy on testset for MobileNet with submission to Kaggle.

3.2.5 Conclusion

In this part, the MobileNet is applied. And in model detection part, batchsize and STEPS are all found positive correlation with the accuracy. Finally, with well-tuned parameters, the MobileNet can reach accuracy about 0.88 on testset.

3.3 RNN

3.3.1 LSTM

We first follow the Tensorflow RNN tutorial [8] to do the classification for Quick, Draw! dataset. As shown in Figure 15, The model takes the stroke data and 'preprocesses' it a bit using 1D convolutions and then uses two stacked LSTMs followed by two dense layers to make the classification. The model can be thought to 'read' the drawing stroke by stroke.
Figure 16 shows the training/validation accuracy as well as the training/validation loss against the number of epochs, using the LSTM modeled described in Figure 15. We did some minor hyperparameter tuning including adjusting the dropout rate for the convolutional layers and adjusting the batchsize. The best MAP3 we can obtain using LSTM is 0.685, which is fairly low compare to the image-based model such as CNN or MobileNet.
Despite the poor performance, our model does provides some understanding to how LSTM is working. For instance, we used trained LSTM model to classify test Quick, Draw! samples stroke by stroke. As shown in Figure 17, prediction accuracy is very low when only a few strokes being fed into our LSTM model, and as the number of strokes increases, the prediction accuracy increases as well. This observation is a clear indication that LSTM is indeed doing stroke-based classification rather than taking the picture.

3.3.2 LSTM+CNN

Despite the relatively poor performance of RNN model we tried in Sec. 3.3.1, we were inspired by this paper [9] to give a shot on CNN-RNN network. As we know, CNN-based network completely abandons the inherent stroke-level temporal information of human sketches, which can now be modeled by a RNN network. The model put forward in [9] combines the best from both parts for human sketches – utilizing CNN to extract abstract visual concepts and RNN to model human sketching temporal order.

Figure 18 shows an illustration of two-branch network. Note the CNN encoder takes in a raster pixel sketch and translate into a high-dimensional space, whereas the RNN encoder takes in a vector sketch and outputs its final time-step state. There is an late-fusion layer concatenate two branches to

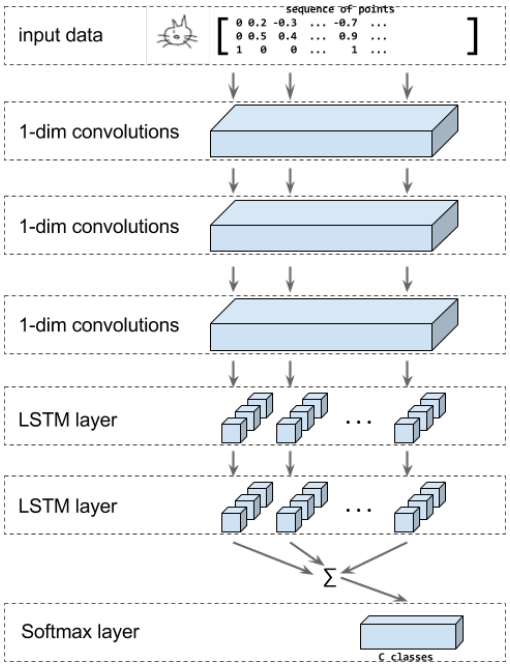

Figure 15: Illustration of stroke-based LSTM model.

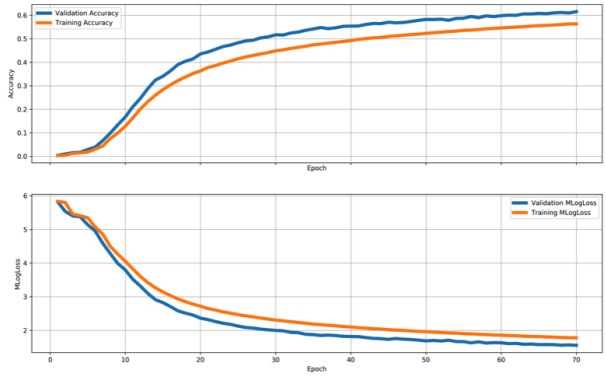

Figure 16: Accuracy and loss against number of epochs, using LSTM MAP3=0.685, which is the best we achieved.

enable our learned feature to benefit from both vector and raster sketch. In practice, we used cross-entropy as loss function (though the original paper proposed much more complicated loss function). For convenience, we followed the suggestion from [10] on building our model and selecting parameters. Figure 19 presents the training / validation accuracy/loss against the number of epochs using the CNN-RNN network. Despite extremely long time in training this two-branch model, the performance is good. The best AP3 (top 3 accuracy) reached 0.865, which is relatively high. It is pity that due to time/resource limitation, we didn't do hyperparameter tuning for our RNN-CNN model, which apparently has potential to be further improved.

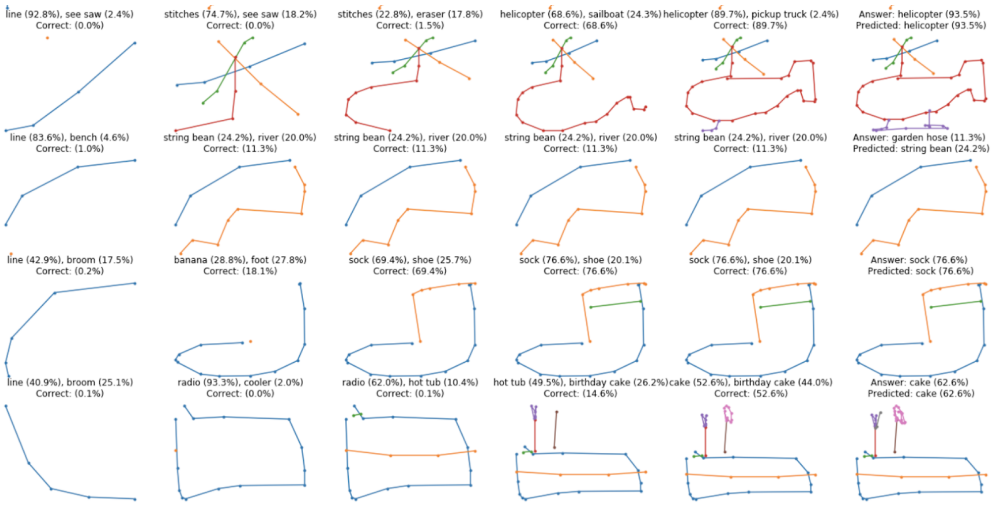

Figure 17: Reading data and predicting stroke by stroke shows LSTM is indeed making classification stroke by stroke.

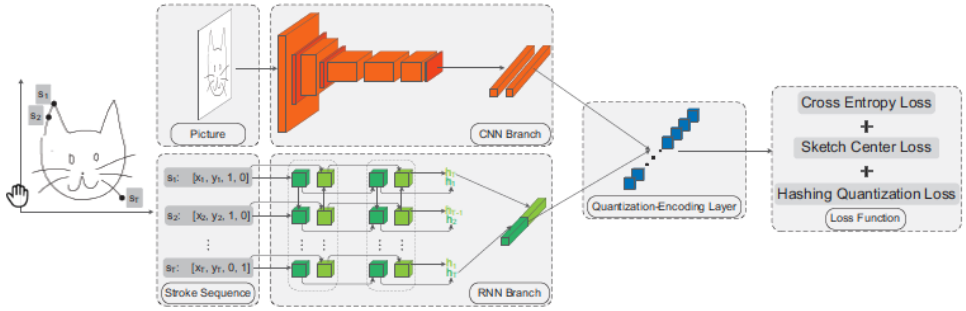

Figure 18: Combining two branches, CNN and LSTM for classification of human sketeches. This illustration originally comes from [9].

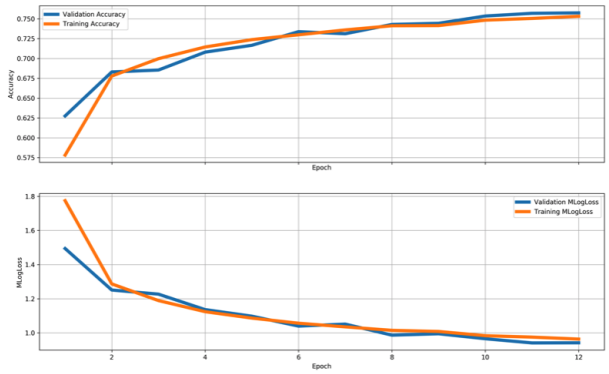

Figure 19: Accuracy and loss against number of epochs, using CNN+LSTM with some hyperparameter tuning, MAP3=0.865.

4 Summary

"Quick, Draw!" was released as an experimental game to educate the public in a playful way about how AI works. The aim of this paper is to build models that can effectively classify the human sketches collected from Quick, Draw! game. In Sec. 3.1 – 3.3, we tried different models, i.e., imaged-based CNN, MobileNet model, stroke-based RNN model and a hybrid CNN-RNN model. While all models turns out to be capable in classification, the performance differs from one to another. According to our experiments, MobileNet gives the best classification performance, indicating that MobileNet is indeed of an effective improvement over CNN. Stroke-based model, i.e. LSTM, turned out to perform less successful as the image-based model. The training expense of LSTM is also found to be significantly larger than MobileNet. Despite all this disadvantages, our analysis reveals the mechanism of LSTM that it is indeed classifying sketches stroke by stroke. We also did some preliminary experiments on a CNN-RNN hybrid model, of which result showed great potential for future investigation.

References

[1] Quick, Draw! The Data,
`https://quickdraw.withgoogle.com/data` (retrieved Dec 4th, 2018)

[2] Quick, Draw! Doodle recognition challenges,
`https://www.kaggle.com/c/quickdraw-doodle-recognition` (retrieved Dec 4th, 2018)

[3] Quick, Draw! dataset github page,
`https://github.com/googlecreativelab/quickdraw-dataset#the-raw-moderated-dataset`
(retrieved Dec 4th, 2018)

[4] Kagge Kernel: Black&White CNN [LB=0.77],
`https://www.kaggle.com/gaborfodor/black-white-cnn-lb-0-77` (retrieved Dec 9th, 2018)

[5] Howard, Andrew G., et al. *"Mobilenets: Efficient convolutional neural networks for mobile vision applications."*. arXiv preprint arXiv:1704.04861 (2017).

[6] Google's MobileNets on the iPhone,
`http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/` (retrieved Dec 9th, 2018)

[7] Kaggle Kernel: Greyscale MobileNet [LB=0.892],
`https://www.kaggle.com/gaborfodor/greyscale-mobilenet-lb-0-892` (retrieved Dec 9th, 2018)

[8] Recurrent Neural Networks,
`https://www.tensorflow.org/tutorials/sequences/recurrent` (retrieved Dec 8th, 2018)

[9] Xu, Peng, Yongye Huang, Tongtong Yuan, Kaiyue Pang, Yi-Zhe Song, Tao Xiang, Timothy M. Hospedales, Zhanyu Ma, and Jun Guo. *"SketchMate: Deep Hashing for Million-Scale Human Sketch Retrieval."*. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8090-8098. 2018.

[10] Combining CNN and RNN,
`https://www.kaggle.com/huyenvyvy/fork-of-combining-cnn-and-rnn/notebook` (retrieved Dec 8th, 2018)