11-785 PROJECT FINAL REPORT: DOODLING IN THE DEEP.

Ramesh Balaji rbalaji
Ernest Yucheng Chang ychang2
Tanmaya Dabral tdabral
Irina Javed irjaved

ABSTRACT

Doodle recognition is the task of identifying an object from the image of its drawing and as such is an N-way classification task. Conventional image classification approaches however do not work as well on doodles as they do on photographs because they learn to discriminate on the basis of spatial features alone. Our idea is to do better than that by also considering the order in which the strokes are drawn. We exploit the dependence of the identity of the object on the temporal ordering of the strokes that make up its drawing. We present our investigations as a series of network training experiments each one building on the previous models. Our initial models tackle the spatial and temporal dependencies independently, whereas the final ones combine them. We further analyze the errors produced by these models and offer insights on the reasons for these errors.

1 INTRODUCTION

Quick, Draw! is an online game developed by Google that challenges players to draw a picture of an object or idea and then uses artificial intelligence to guess what the drawings represent. The AI learns from each drawing, increasing its ability to guess correctly in the future. The game is similar to Pictionary in that the player only has a limited time to draw (20 seconds). The concepts that it guesses can be simple, like 'purse', or more complicated, like 'Eiffel Tower' or 'Animal Migration'. The task at hand is to build a recognition system that, given the doodle, can classify it into one of 340 categories [7].

One of the issues is that since the training data comes from the game itself, drawings can be

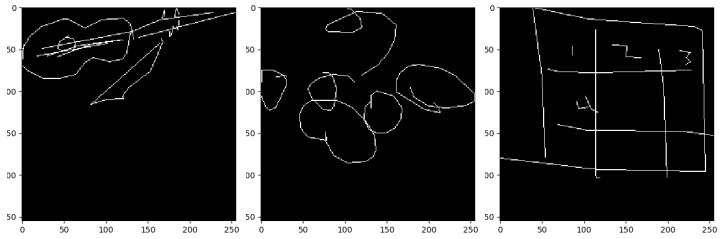

Figure 1: Some images from the dataset demonstrating the noise. The images are labeled violin, peas and calendar respectively. The strokes were rendered using OpenCV.

incomplete or may not match the label. Therefore, the developed recognition system should be able to effectively learn from this noisy data and perform well on a manually-labeled test set from a different distribution. Moreover, the size of the dataset is immense and it contains about 50 million data-points. The dataset from Kaggle provides two representations for each doodle. In both of them, the doodles are represented as a series of strokes. However, in the raw version of the dataset, each stroke is a list of all the points in the stroke while in the simplified version, each stroke is represented as a list of anchor points for an approximate polyline, generated by running the Ramer-Douglas-Peucker algorithm[5] on the raw stroke. As can be seen, the data contains temporal information about the strokes apart from the pictorial information typical to any image recognition task. We seek to build a classifier for the existing Quick, Draw! Dataset, by learning a representation that leverages this temporal order of the strokes as well as their relative positioning in two dimensional space. We hypothesize that a network that captures both the temporal nature of the strokes using a Recurrent Neural Network, as well as the pictorial nature of the drawing using a

Convolutional Neural Network (CNN), will perform better than either of these techniques alone.

2 Related Work

[1] provides a performance benchmark for many popular CNN architectures, allowing us to pick a relatively computationally inexpensive option. Given the enormous size of the dataset, balancing performance and computational complexity is a major challenge.

[6] introduces Logit Boosting, a method to build ensembles of multiple simple models which uses a weighted sum of the logits produced by the individual models. Our best performing model uses a similar approach.

[7] is an online challenge where we are provided with the training data along with the corresponding labels, as well as the testing data without the corresponding labels. We are supposed to predict the top 3 labels for the test data and submit the predictions. An online judge calculates the MAP@3 score for our submission and places us on a leaderboard.

[9] introduces the concept of residual learning, in which the layers are trained to learn a residual function with respect to the layer input. Our best performing CNN closely follows an the 18-layered ResNet architecture described in the paper.

[15] describes MobileNetV2, which builds on the previously introduced MobileNet [11], a computationally efficient CNN, which uses depth-wise separable convolution. We have implemented this model in PyTorch and use it as a baseline.

[17] describes an LSTM based model for this specific task. The model performs 1D convolutions on the input, and feeds the convolved inputs to an LSTM unit, which then performs the classification. We have implemented this model in PyTorch and use it as a baseline.

[18] provides a lucid explanation of depth-wise separable convolutions and we have used it as a reference for the implementation.

3 Methods

3.1 Data Preprocessing

As mentioned in the previous sections, we try two distinct types of data representations, one capturing the temporal information available, and the other capturing the spatial information We also try a third representation that tries to capture the available temporal information in pictorial form. All these representations require different preprocessing steps:

3.1.1 Sequence-of-Points Representation

To make the data format usable with a Recurrent Neural Network(RNN), all the strokes in a doodle are concatenated into a single sequence of anchor points. However, to preserve the end-of-stroke information, we add a third, binary dimension to each point, which is 1 if the point is the beginning of a new stroke and 0 otherwise. Furthermore, for each doodle, we normalize the axis with the larger range to [0, 1] and scale the other axis to maintain the aspect ratio, and then translate the points so that the doodle is centered at (0.5, 0.5).

3.1.2 Pictorial Representation

The GitHub repository for the Quick, Draw! dataset also provides the doodles in a 28x28 pictorial format, rendered from simplified strokes. However, to gain a finer control on the resolution of the images, we render the strokes ourselves instead of using the images provided. We use OpenCV [2], an image processing library, to render the images on the fly batch-wise. To cut down on this significant overhead, we spawn a parallel process to generate the images and maintain a buffer of multiple batches. We try two different image resolutions: 28x28 for the baseline and 224x224 for the

more complex models. The lines in the 28x28 images are ant-aliased to preserve more information than a binary image would.

3.1.3 Time-coded Pictorial Representation

In addition to the spatial information inherently present in images, we also try to encode temporal information into the images by varying the pixel intensities for each stroke based on the temporal order. Specifically, the pixel value for the first stroke starts from 255 and it is linearly decreased by 13 for each stroke until it reaches 125. The underlying hypothesis is that a human is likely to draw a doodle by starting with the most prominent features of the object, and the strokes in the beginning may therefore be critical to determine the class of the object drawn.

3.2 Models

We try six different models, each with a different complexity. We train the simpler models on the entirety of the dataset and the more complex models on only a part of the dataset. All models are trained using categorical cross-entropy as the loss function and ADAM [14] as the optimizer. We use the benchmarks provided by [1] to choose CNN architectures.

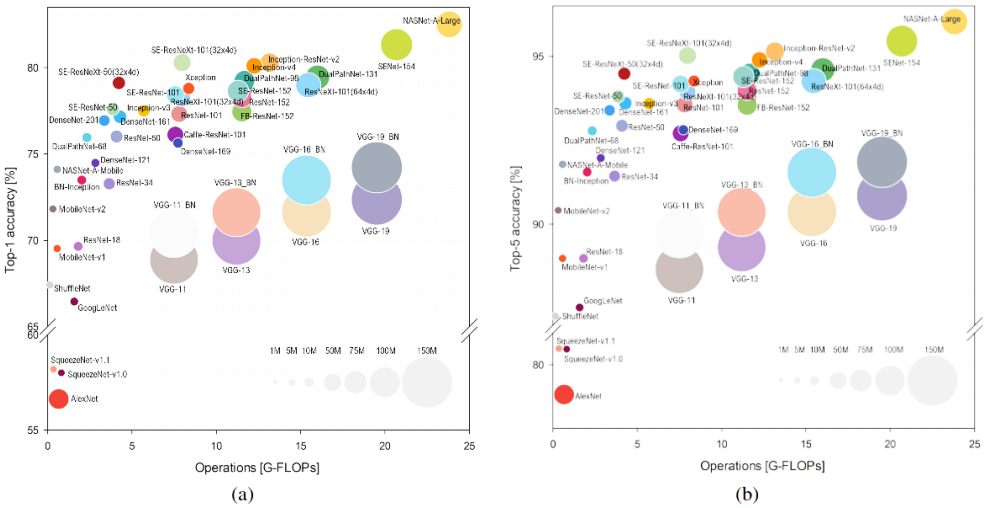

Figure 2: Ball Chart representing the Top-1 and Top-5 accuracy vs. computational complexity. Top-1 and Top-5 accuracy using only the center of the crop versus floating-point operations (FLOPs) required for a single forward pass are reported. The size of each ball corresponds to to the model complexity. (a) Top-1 ; (b) Top-5. (Source: Bianco et al.[1])

3.2.1 Conv1D - LSTM-128 (Sequence-of-Points Representation)

This model, borrowed from [17] is composed of two distinct parts. The model first runs a series of three 1D convolutions along the time dimension on the sequence of points. The convolution filter sizes are 48, 64 and 96. Each of these convolutional layers is activated using the ELU activation function [4] and is followed by a batch normalization layer [12]. The output of the convolutional layers is then fed to a three layered Long Short-Term Memory (LSTM) unit [10] with 128 hidden units. The initial hidden states are trainable tensors. The output of the LSTM is averaged along the time dimension and fed to a fully-connected linear layer of size 340, yielding the logits for each class. This model serves as the baseline for the Sequence-of-Points classifiers.

3.2.2 MobileNetV2 (Pictorial Representation, 28x28)

As our second baseline, we use MobileNetV2 [15], a computationally efficient CNN. The MobileNetV2 uses two tricks to decrease the number of computations, when compared to a vanilla

CNN. First, it factorizes a standard convolution into a depth-wise convolution (where each channel is convolved independently) followed by a standard 1x1 convolution which combines the outputs of all the channels. This operation is much cheaper computationally than running a standard convolution. Secondly, the architecture adds on to this efficiency by introducing inverted residuals and linear bottlenecks. We train this network with 28x28 images as inputs.

3.2.3 CONV1D - BILSTM-512 (SEQUENCE-OF-POINTS REPRESENTATION)

This model is a more complex version of the Sequence-of-Points baseline. The 1D convolutional layers remain the same while the LSTM is replaced with a Bidirectional LSTM [16] with 512 hidden units in each direction. Once again, the output of the BiLSTM is averaged along the time dimension and fed to a linear layer to obtain the logits.

3.2.4 RESNET-18 (PICTORIAL REPRESENTATION, 224X224)

This model closely follows ResNet-18 described [9], which introduces residual blocks for easier flow of gradients in deeper architectures. Specifically, the architecture is a stack of a basic CNN block followed by four residual CNN blocks, totalling to 18 layers. Given the representation power of the network, we train it on 224x224 images. We initialize the network with weights trained on the ImageNet dataset.

3.2.5 RESNET-18 (TIME-CODED PICTORIAL REPRESENATATION, 224X224)

The architecture for this model is exactly the same as the previous model (3.2.4). However, instead of using binary images, we use the time-coded pictorial representation described previously.

3.2.6 RESNET-18 + CONV1D - BILSTM-512 LOGIT ADDITION

Before implementing an ensemble, we evaluated a simple addition of the logits yielded by the trained ResNet-18 (3.2.4) and the trained Conv1D -BiLSTM-512 (3.2.3), which yielded surprisingly good results.

3.2.7 RESNET-18 + CONV1D - BILSTM-512 ENSEMBLE

Finally, we combine the trained ResNet-18 (3.2.4) and the Conv1D - BiLSTM-512 (3.2.3) into a Logit Boosted [6] ensemble, where we derive the final logits as a weighted combination of the logits produced by each individual model. The weights, along with the parameters of each model are trainable.

4 RESULTS

To evaluate our models we employ MAP@3(Mean Average Precision) as the performance metric, which takes into account the top 3 predictions instead of just the topmost one.

$$MAP@3 = \frac{1}{U} \sum_{u=1}^{U} \sum_{k=1}^{min(n,3)} P(k)$$

where U is the number of scored drawings in the test data, $P(k)$ is the precision at cutoff k, and n is the number predictions per drawing. Given the noise in the dataset and the large number of classes, a metric more lenient than accuracy makes sense. We present the scores obtained for each model, along with the fraction of data used for training the models, in table 1. We analyze these results in section 5.

5 DISCUSSION

One apparent pattern that we find is that increasing the complexity of the model yields significant gains in the MAP@3 scores, both for sequence-of-points and pictorial representations. This is further corroborated by the discussions on the Kaggle forums, where it has been shown that more complicated architectures like DenseNet-201, ResNet-152, SE-ResNet-101 and SENet-154 have been

Model	Fraction of training data used	Training time in hours	MAP@3
Conv1D - LSTM-128 (stroke)	1.0	17	0.804
MobileNetV2 (28x28)	1.0	12	0.819
Conv1D - BiLSTM-512	0.1	14	0.864
ResNet (224x224)	0.1	16	0.870
Time-coded ResNet (224x224)	0.1	16	0.889
ResNet + Conv1D - BiLSTM-512 logit addition	-	-	0.905
ResNet + Conv1D - BiLSTM-512 Ensemble	0.1	21	**0.917**

Table 1: Performance of each classifier on the test data.

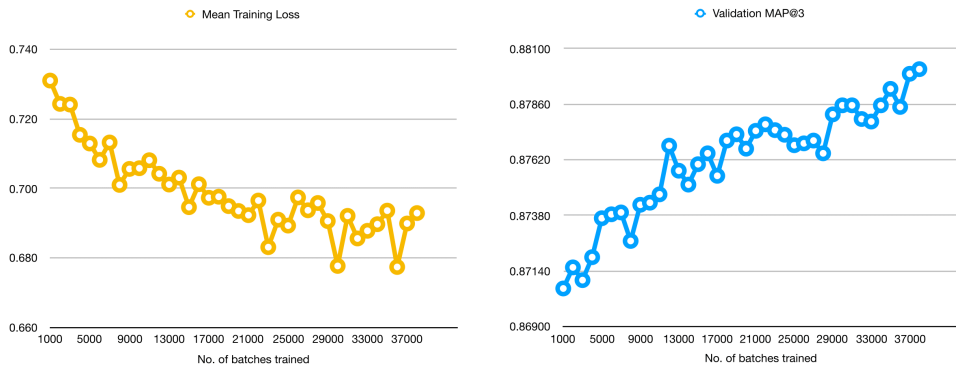

Figure 3: Training Plots for the **ResNet + Conv1D - BiLSTM-512 Ensemble** classifier. The left plot shows the training loss over time. The right plot shows the validation MAP@3 over time. Note that the validation MAP@3 begins from a high value because this is an ensemble of classifiers that are already trained. Also note, that the the MAP@3 values on the validation set are lower than those on the test set. This is due to the fact that the test set has cleaner doodles (read Section 5).

able to achieve MAP@3 scores of upto 0.946 [13]. However, we stick to relatively simpler models owing to the scarcity of computational resources.

Furthermore, our hypothesis that using both the temporal and spatial representations together should yield better performance is validated by the following observations:

- The performance of the time-coded ResNet is significantly better than the performance of the vanilla ResNet.
- The BiLSTM-ResNet ensemble performs better than either of the two models individually.

What is surprising, however, is that even a simple addition of logits of two models performs better than each individually. We believe that this is possible because both the models learn to output logits that are similar in magnitude, so that neither dominates the other.

We must also note here that, as is evident from the graphs, the ensemble had not converged fully and training on more data would have possibly yielded better results. However, for the sake of comparability, we stopped the training preemptively.

Another interesting observation is that the test MAP@3 scores (obtained from the Kaggle public leaderboard) are higher than the MAP@3 scores obtained on the validation dataset. This is, however, expected, since the test dataset is hand-crafted and so has much lesser noise than the train dataset, from which the validation dataset is derived.

6 FUTURE WORK

An interesting direction for future work is to find a more intelligent way to time-code the strokes fed into CNN. Future work can the explore various ways to decrease the intensity of the pixel values after a certain number of strokes. We can measure the effect of decaying the pixel value exponentially on the learning abilities of the network. We can even tune the decay rate separately for each class based on the fact that they belong to different distributions. What's more, this method has potential to de-noise our dataset by dealing with meaningless scribbles by assigning the pixel values of strokes above certain number to zero.

As specified in our discussions section 5, complicated CNNs with large number of parameters perform well on this dataset. Based on relevant literature, the model performance of NASNet-A-Large is the best and it has potential to outperform these aforementioned models. Therefore, it's worthy of implementation.

Lastly, we also propose different ensemble methods for future work as follows. The first, simpler approach is to use a CNN-based feature extractor that extracts features from the entire image. We flatten these 2D feature maps and append the entire vector to each coordinate that is fed to the recurrent unit. We hope that the LSTM learns to focus on the relevant parts of these 2D feature maps for each time step. We intend to pre-train the CNN for the entire task and then select one of the hidden convolutional layers and use the output of that layer as our feature embedding. Also, the second approach is something akin to what has been used for visual question answering in [3]. Here, a convolution kernel is dynamically generated at the final time-step by the RNN unit, and the feature-map extracted by the CNN is convolved with it. Finally, the original feature map, the newly generated feature map, and the RNN output are concatenated and fed to a classifier. We suspect that such an approach, where the convolution filter is dynamically generated, might be computationally expensive. Therefore, we may try to simplify the model by generating a probability distribution over all the pixels of the feature map, which we will then use to weight the pixels before feeding everything to a classifier.

One can also investigate the generative task of generating doodles conditioned on a label. One can use our improved representation of the data and given the nature of the data, borrow from language generation techniques, as done in [8] and apply it to the task of doodle generation.

REFERENCES

[1] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 2018.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Kan Chen, Jiang Wang, Liang-Chieh Chen, Haoyuan Gao, Wei Xu, and Ram Nevatia. Abc-cnn: An attention based convolutional neural network for visual question answering. *arXiv preprint arXiv:1511.05960*, 2015.

[4] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[5] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[6] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[7] Google . Quick draw! challenge, kaggle competitions. `https://www.kaggle.com/c/quickdraw-doodle-recognition/data`.

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

[9] 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*, pp. 770–778, 2016.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[11] Andrew 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.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[13] Kaggle. Quick draw! challenge, kaggle competitions. `https://www.kaggle.com/c/quickdraw-doodle-recognition/discussion`.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *arXiv preprint arXiv:1801.04381*, 2018.

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

[17] Tensorflow . A tutorial for recurrent quickdraw. `https://www.tensorflow.org/tutorials/sequences/recurrent_quickdraw`.

[18] Towards Data Science . An introduction to different types of convolutions in deep learning. `https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d`.