# PREDICTING SKETCHES THROUGH CONVOLUTIONAL NEURAL NETWORKS

Thomas Patton
Case Western Reserve University
Department of Computer Science

## I. Introduction

The act of image classification has been a long-standing benchmark of artificial intelligence. From the first neocognitron to the advanced networks that exist today, the ability of machines to intelligently recognize images has improved to be nearly indistinguishable from that of a human. This incredible capability was the motivation behind this project: Knowing that networks can classify images incredibly well, what performance could we expect to see not on images themselves but instead sketches of *concepts* of images? More specifically, how well could a network be trained to recognize user created sketches of image classes? In this paper, I want to investigate the use of convolutional neural networks on Google's *Quick, Draw!* dataset to make the conclusion that even a simple network can recognize quick sketches with surprising accuracy.

## II. Materials

As was previously stated, Google's *Quick, Draw!* dataset is the focus of this project. This dataset contains 50 million drawings across 345 object classes which were contributed by human players playing Google's game *Quick, Draw!*. This game prompts users to draw one of the 345 object classes within a 20 second timer and thus captures a user's *sketch* of a concept class. Examples of these sketches are shown in *Fig. 1*.

345 concept classes seemed a bit ambitious for this project, however, and so the scope had to be narrowed. The goal for this project was to accurately categorize images belonging to 10 classes: *Bear, Bee, Bird, Cat, Dog, Dolphin, Fish, Horse, Shark* and *Tiger*. I picked these classes because they all pertained to a common theme of animals. I also intentionally picked some classes which I knew would be easy to distinguish from each other (i.e. "Bear", "Fish") and others that would be difficult (i.e "Cat", "Tiger").

On the Github page for GoogleCreativeLab, these drawings are easily accessible. I chose to download each as static NumPy bitmaps, though vectorized versions of the sketches (i.e. time-dependent *strokes*) are availible. Classifying these dynamic images with a recurrent neural network is a possible future direction for this project.

Fig. 1: Example sketches belonging to the classes "Bear", "Bird", and "Tiger" respectively

Lastly, for this project, TensorFlow version 2.3.1 was used in conjunction with Python version 3.7.6.

## III. Methods and Experiments

The purpose of this section is to detail my process of attempting to find the optimal network design. To do a complete analysis of every configuration would be beyond the scope of this project and would be nearly impossible to read. Instead small tweaks were made to different hyperparameters based on previous results in an attempt to best tune them. The network was initially based off the convolutional net used in Google's "Cat vs. Dogs" example.

### A. Pre-Processing

To start, I wrote some pre-processing code which took in the NumPy bitmap files, turned them into grayscale images, and then saved them as .png files in the appropriate images directory for training. I limited my code to only taking 5,000 of each 28x28 image and used an 80/20 split for train/test.

### B. Batch Size

My network began with a batch size of 10 which was completely inadequate for the problem and the validation and training accuracy both topped at around 0.71 after 5 epochs. This batch size was adjusted to 100 which gave good improvement to the training accuracy (0.83) but did not affect the validation accuracy. Increasing the batch size to 1,000 started to cause a decrease in performance and so it was moved back to 100.

### C. Optimizer

This project was initially set to optimize using Binary Crossentropy loss with RMSprop and a learning rate of 0.0001. I altered this to be a Categorical Crossentropy loss function (because of the multiple categories) and changed the optimizer to Adam. However, this didn't seem to have a real effect on performance - both models topped out with a validation accuracy of around 0.7 and a training accuracy of around 0.85.

### D. Padding

When working with smaller images like this the choice of padding for the convolutions can actually be extremely important. Setting the padding configuration on my Conv2D layers to be 'same' gave a large boost in performance with the training set reaching an accuracy of 0.93 and the validation set reaching an accuracy of 0.75.

### E. Dropout

Adding dropout turned out to be another major gain for the network. By adding a dropout of 0.5 after the convolutional layers the network was able to achieve a training accuracy of 0.79 and a validation accuracy of 0.78. We can also see that overfitting has been drastically reduced here, as our training and validation accuracies are very close.

### F. Conclusion

Beyond what is listed here, there were many additional experiments performed. Network architecture, learning rate, shuffling, and other parameters were tuned. These listed changes were those which had a significant impact on performance. For many experiments the results were either identical to before or crashed convergence altogether. In total over 100 modifications to the network were tested.

## IV. Results

After these experiments, a final "optimal" model was reached. This model's predictions on the validation set were obtained after 30 epochs and the results of ScikitLearn's classification report are displayed in *Table I*. TensorFlow reported this model to have an accuracy of *0.7996*.

TABLE I: Results

| | Precision | Recall | F1 Score |
|---|---|---|---|
| bear | 0.38 | 0.52 | 0.52 |
| bee | 0.91 | 0.85 | 0.88 |
| bird | 0.82 | 0.78 | 0.80 |
| cat | 0.85 | 0.70 | 0.77 |
| dog | 0.69 | 0.53 | 0.60 |
| dolphin | 0.88 | 0.66 | 0.76 |
| fish | 0.93 | 0.87 | 0.90 |
| horse | 0.87 | 0.73 | 0.79 |
| shark | 0.83 | 0.80 | 0.81 |
| tiger | 0.79 | 0.70 | 0.75 |

### A. Class Variability

The biggest takeaway from this table is that the performance over each animal class varies quite widely. "Fish" produced the best F1 score with 0.90 while "Bear" had an F1 score of just 0.52. The model's guess about "Fish" is near certain while its guess about "Bear" is more akin to a coin flip. Delving a little deeper into the problem, I realized why this was: the drawings for the class "bear" were far less coherent and uniform than those for "fish".

Intuitively, though, this makes sense - with only 20 seconds to draw, humans must quickly *symbolize* ideas about the concept class. *Symbolize*, here, means an easy and straightforward representation of a given class (e.g. a "Clock" being a circle with two internal arrows). Going back to the original *Quick, Draw!* dataset, those classes which can be easily symbolized ("Apple", "Baseball", "Cloud", "Fish", "Mountain", etc.) have nearly ubiquitous answers. But returning to our "Bear" example, even the best drawings capture bears from many different angles and poses as there isn't an easy symbolic representation of a bear. In addition, many drawings for these more complex classes are either incomplete, crudely sketched, or are scribbled out. The average human wouldn't be able to identify many of these sketches as bears and so it isn't surprising that our convolutional net cannot either. In retrospect, the accuracy of this model is deceptively high.

### B. Object Similarity

As was mentioned earlier, some classes were chosen on the premise that they would be tough to distinguish. ("Cat", "Tiger"), ("Bee", "Tiger"), and ("Dolphin", "Shark") are all examples of this as they share the properties *cats, stripes,* and *fins* respectively. To investigate how this ultimately manifested, TensorFlow's confusion matrix was used. Here the matrix columns

TABLE II: Confusion Matrix

| | bea | bee | bir | cat | dog | dol | fis | hor | sha | tig |
|---|---|---|---|---|---|---|---|---|---|---|
| bea | 848 | 7 | 9 | 24 | 58 | 1 | 6 | 23 | 1 | 13 |
| bee | 83 | 857 | 11 | 6 | 3 | 2 | 1 | 1 | 3 | 33 |
| bir | 197 | 12 | 719 | 4 | 15 | 26 | 3 | 3 | 14 | 7 |
| cat | 199 | 4 | 1 | 712 | 36 | 0 | 1 | 10 | 3 | 34 |
| dog | 290 | 5 | 17 | 40 | 508 | 2 | 8 | 106 | 7 | 17 |
| dol | 130 | 2 | 22 | 4 | 4 | 680 | 17 | 3 | 132 | 6 |
| fis | 65 | 3 | 4 | 2 | 3 | 5 | 877 | 0 | 35 | 6 |
| hor | 119 | 2 | 8 | 2 | 45 | 9 | 1 | 792 | 1 | 21 |
| sha | 95 | 5 | 4 | 4 | 1 | 45 | 29 | 2 | 813 | 2 |
| tig | 204 | 27 | 3 | 55 | 30 | 1 | 0 | 18 | 2 | 660 |

represent the predicted labels while the rows represent the true labels. The results are in *Table II*.

Investigating our queries:

- Our model predicted "Cat" for "Tiger" 55 times and predicted "Tiger" for "Cat" 34 times.
- Our model predicted "Bee" for "Tiger" 27 times and predicted "Tiger" for "Bee" 33 times.
- Our model predicted "Dolphin" for "Shark" 45 times and predicted "Shark" for "Dolphin" 132 times.

While these numbers aren't overly high, they are some of the largest entries of this matrix that don't lie on the diagonal. Other high entries include mistaking "Dogs" for "Bears", mistaking "Horses" for "Dogs" and "Fish" for "Sharks". The bigger observation, however, lies in the "Bear" column. We can see the model drew a majority of its error from mistaking things for "Bears". This strengthens the idea that the drawings for "Bear" were worse than the other classes. With the model unable to gain a clear idea of what a "Bear" was, any drawing which was too unclear got lumped into the category of "Bear".

## V. Conclusion

The performance of this network indicates that convolutional networks are able to recognize sketches of concept classes with surprisngly good accuracy. In more general sense, this project shows that convolutional nets are more than capable of learning abstractions - an extremely powerful tool with a wide range of influence. However, classes which are difficult to easily symbolize create poor training data and can negatively impact network performance.

In terms of future directions for this project there are a few ideas worth mentioning. As was previously mentioned, Google offers vectorized versions of these sketches and using a recurrent neural network one could create a "real-time" sketch predictor which changes its answer as a user inputs a sketch. With more computing power, one could attempt to generalize this network to handle more object classes. Lastly, one could attempt to develop a metric to measure "image distance" to determine which concept classes have the best chance for generalization.

In conclusion, this problem as a whole was extremely fun to implement and explore. There were many concepts related to convolutional networks, categorical classification, and training data that I learned specifically because of the uniqueness of this project. I look foward to taking this knowledge and applying it to all projects I do in the future.