Sketch Classification

Chandler Tayek
{chandlertayek@gmail.com}

Abstract

Sketch classification has been growing in popularity ever since Google published their data-set QuickDraw back in 2012. The idea of sketch classification is to take a hand drawn doodle and correctly select the most likely category it belongs to. In this paper I present a convolutional neural network based off the Guo et al. model. The altered model achieves a 92% accuracy on 15,000 hold-out test images.

1. Introduction

Sketch classification is a challenging task due to high levels of abstraction and the fact that modern convolutional neural networks (CNNs) mostly learn surface statistic regularities [3]. A robust sketch recognition approach can empower many practical downstream applications, such as graphical design, sketch-based retrieval or modeling. My proposal investigates the effects of tuning a convolutional neural network on a multi-class supervised learning problem. To do so, I will base a convolutional neural network off the Guo et al. model [1]. In the sections below I will explain the current field, information about the data-set, the model's architecture, and some discussion on the results.

2. Related Work

A good portion of the successful doodle classification models use some variant of a neural network. This isn't that surprising especially after the success that the LeNet[4] architecture had with classifying handwritten digits. LeChun et al. showed that having several convolutional filters can obtain different levels of abstracted information about the image and work just as well as it would in a less noisy problem space. SketchNet: [8] was another model that used convolutional layers but in this case, to classify sketches. Zhang et al. showed that ensemble feature representations are state of the art in CNNs. Their model out preformed every major model at the time, besting FisherVectors[5], GoogleLeNet, and even AlexNet.

Although CNNs seem to have been dominating the field at the time, recurrent neural networks(RNNs) proved to be the next best thing. RNNs function by looking at windows of time. This lets RNNs get at information such as stroke data[2] that the CNN models lacked access to. This stroke data helps to give the model a since of the early, middle, and end stages of a doodle rather than just a static image. Information form stroke data showed to be a powerful feature and even powerful enough to beat human level classification[7]. RNNs are not the only model that can build around stroke data. Yu et al. model is a neural network that is based off time windows. There has been some other interesting proposals on stroke and corner detection such as Shpitalni and Lipson's work[6].

It is worth mentioning a project by the user Payalbajaj located on GitHub [1]. They use the same QuickDraw data-set and almost the same structure as Google's version of an RNN with some slight modifications such as changing the loss to a cross-entropy which happens to be the same loss function that I'm using. Payalbajaj's model achieves a 92% on sixteen classes.

3. Data

3.1. Acquisition

The data was obtained from an open source project from Google called QuickDraw[2]. QuickDraw is a game, designed to capture lots of manually drawn sketches in order to explore 2D mono-color sketch classification. Google offers the data-set in several different formats including numpy bitmaps which I selected to use for my pipeline.

Google offers the data via a cloud storage system[3] where each file is an Nx784 bitmap that can be loaded directly into a numpy array.

4. Exploration

QuickDraw contains 50 million sketches covering 345 classes. The data-set has already been prepossessed by the repository owner. Figure 1 shows a random example from each of the ten categories used by my model. It is important

[1] https://github.com/payalbajaj/sketch$_r nn_c lassification$

[2] github.com/googlecreativelab/quickdraw-dataset

[3] https://console.cloud.google.com/storage/browser/quickdraw$_d ataset/full/numpy_b itm$

1

to note that since the data-set can be added to by anyone that as an internet connection, some drawings are so off that a human wouldn't even be able to correctly classify it. This causes there to be a significant amount of noise added into the data-set. As you can see in figure 1, the piano could easily be classified as a radio and vice versa.

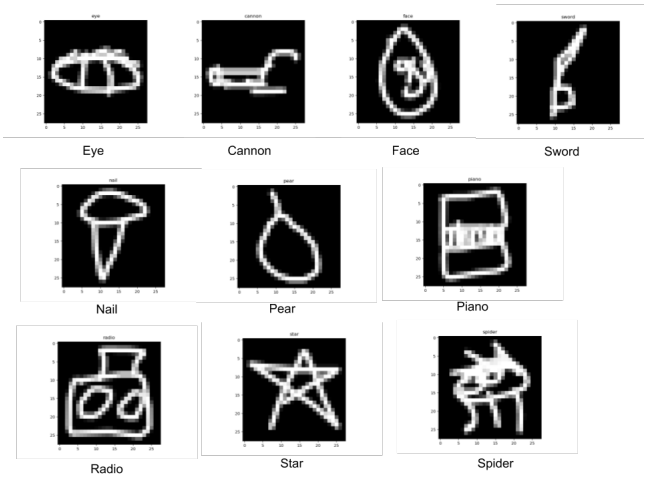

Figure 1. examples of the different classes

5. Methods

5.1. Sketch classification model

A subset of 10 categories were randomly selected from the available 345 {Eye, Cannon, Face, Sword, Nail, Pear, Piano, Radio, Star, Spider}The data-set was split into three different sections: training, validation, and testing. Both the validation and the test set comprised of 15% of the total sample each. The test set was put to the side and only used once at the very end once the final model was prepared.

For the classification model, I based it off the Guo et al. model. Each image was normalized by subtracting the mean then dividing by the standard deviation over every example. A small epsilon was added to the standard deviation to account for division by zero. Both the validation and test set were normalized with the same mean and standard deviation from the training set.

Figure 2 shows the entire architecture for the model. The first layer takes n 28x28x1 images and then passes them on to three convolutional filters, each with a 3x3 filter and five output channels. The next part is a max pooling layer with kernel size 2x2. AFter that comes a fully connected layer that feeds into 8 dense layers with {700, 500, 400, 300, 200, 100, 50, 10} units respectfully. Each dense layer is activated with a relu function and then passed through a dropout layer. The last dense layer uses a softmax instead of a relu.

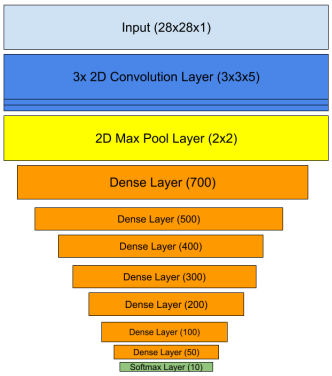

Figure 2. model architecture

5.2. Evaluation

The model was trained using cross-entropy loss:

$$-\sum_{n}\sum_{i} y_i^{(n)} log(\hat{y}_i^{(n)})$$

During training, both the loss and the accuracy metric were computed at each epoch. These metrics were very important in order to see how well the model was preforming during parameter tuning.

6. Results

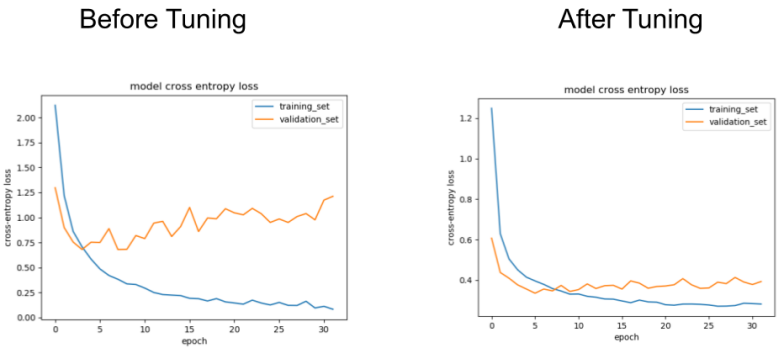

Figure 3. the left is the fourth tuning iteration and on the right is the final tuning iteration.

The initial hyper-parameter settings were {samples: 1,000, dropout: 20%, optimizer: adam}. These produced a training and validation loss of 0.07885 and 1.506 respectfully. The graph produced from this is shown on the left of figure 3. The orange curve represents the validation loss and the blue curve the training loss.

The final iteration of the model ended up training on 7,000 samples from each category totalling 70,000 training examples on 32 epochs. The optimizer that preformed the best was adam. A dropout rate of 40% was the found to be

Accuracy	Value
My Training	93.3%
Guo et al. Training	52.9%
My Validation	92.1%
Guo et al. Validation	53.5%
My Test	91.6%
Guo et al. Test	53.4%

Table 1. my model's accuracy on 10 categories verses theirs on 345 categories

optimal. These parameters produced a training and validation loss of 0.282 and 0.393 respectfully. Our model ended up having a much higher accuracy of 91.6% than the Guo et al. model 53.4%, as shown in table 1. This comes to no surprise though as guessing a single category over ten is a much simpler task than on all 354.

6.1. Tuning

The drop out rate was the most productive knob to tune. If you notice in figure 3, the graph on the left has a much larger separation between the validation curve and the training curve. This was a sign that the model was overfitting. The dropout rate was set to 20%, the same as the Guo et al. model. As I increased the dropout, rate the model seemed to generalize better but going up to 50% caused the model to start to experience high validation loss just as before. The sweet spot was found to be at 40% with its results shown in the right graph of figure 3.

7. Conclusion

Model tuning can be a tedious task and is hard to realize when enough tuning has been completed. My proposed model has shown how having a training loss decrease over each epoch with a much higher validation loss can be a sign of overfitting. By fine tuning the dropout parameter, it showed that adding more regularization helped close the gab between the two. The model became more generalizable over several iterations.

7.1. Future Work

Further improvements on this work could be modifying my model to classify all 345 categories in order to make a more reasonable comparison to the Guo et al. model. This will most defiantly lead to a loss in accuracy. Another avenue worth pursuing would be the RNN route and possibly making an ensemble of RNNs and some CNNs.

References

[1] Kristine Guo, James WoMa, and Eric Xu. Quick, draw! doodle recognition, 2018. 1

[2] David Ha and Douglas Eck. A Neural Representation of Sketch Drawings. *arXiv e-prints*, page arXiv:1704.03477, Apr 2017. 1

[3] Jason Jo and Yoshua Bengio. Measuring the tendency of cnns to learn surface statistical regularities. *arXiv preprint arXiv:1711.11561*, 2017. 1

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. 1

[5] Roslia Schneider and Tinne Tuytelaars. Sketch classification and classification-driven analysis using fisher vectors. 1

[6] M. Shpitalni and H. Lipson. Classification of sketch strokes and corner detection ... 1

[7] Qian Yu, Yongxin Yang, Yi-Zhe Song, Tao Xiang, and Timothy Hospedales. Sketch-a-Net that Beats Humans. *arXiv e-prints*, page arXiv:1501.07873, Jan 2015. 1

[8] Hua Zhang, Si Liu, Changqing Zhang, Wenqi Ren, Rui Wang, and Xiaochun Cao. Sketchnet: Sketch classification with web images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1105–1113, 2016. 1