W4995 Project Final Report – Sketch Classification with Neural Architecture Search and Data Augmentation

Ruisi Wang
Columbia University
116th St & Broadway, New York, NY 10027
rw2720@columbia.edu

Gareth Ari Aye
Columbia University
116th St & Broadway, New York, NY 10027
gga2107@columbia.edu

Tao Li
Columbia University
116th St & Broadway, New York, NY 10027
tl2873@columbia.edu

1. Introduction

We experimented with image classification on Google's "QuickDraw" sketch dataset. Our goals were to observe the performance of an automated learning system on sketch classification and to see if we could augment the base dataset with additional, realistic sketches. Ultimately, we were able to use Google Cloud's AutoML service to train a binary classification model for each class with high performance (average 98.66 AUC on our validation data) and generated more training data through GAN and geometric as well as stroke-based techniques.

2. Overview of the data

We used Google's QuickDraw sketch dataset comprised of 50 million images across 340 categories like "basketball", "horse", and "zigzag". The sketches were taken directly from Google's online pictionary-like game, so there is significant noise. As an example, the sketch below (Figure 1) is labeled as depicting an ambulance.

Figure 1. Example junk image (labelled "ambulance")

Significant overlap among the 340 classes was another challenging aspect of this dataset. Our models needed to differentiate between users' sketches of prompts like "bird", "duck", and "flamingo"; "cup", "mug", and "coffee cup"; and "cake" and "birthday cake".

Although strokes with timing information were available, we made a choice early on to focus primarily on rasterized, grayscale thumbnails. The transformation from strokes with timing information produced 28×28 grayscale bitmaps and included

- Aligning sketches to the center of the drawing's bounding box.
- Uniformly scaling drawings to have maximum values of 255.
- Resampling strokes with 1 pixel spacing.
- Simplifying strokes using the Ramer-Douglas-Peucker algorithm[7] with an epsilon of 2.0.

Figure 2. Correctly predicted owls (top); Correctly predicted non-owls (bottom).

3. Building a Baseline Model with AutoML

Google Cloud offers a novel automated image classification service based on neural architecture search[9]. We adopted AutoML early on so that we could focus on data augmentation and because we were impressed by its accuracy in binary classification for each of the sketch categories (avg. 98.66 AUC).

Neural architecture search is an active research area in meta learning. AutoML's implementation trains a recurrent neural network to generate appropriate child neural networks via reinforcement learning[4]. The AutoML Vision service allows users to upload labelled training examples and uses such an RNN to build an applicable classification network automatically.

Since AutoML limits the number of training examples to one million, we decided to apply a one-vs-rest scheme so that we could leverage all of the original 50 million labeled examples and have the ability to augment the dataset further. One-vs-rest is a popular strategy to combine a collection of binary classifiers together to build multiclass classifications. Using OvR, we were able to train 340 classifiers, providing each with around 300k images. Figure 3 are some of the training images classified by our "owl" category model.

4. AutoML Performance

We include below a table showing the performance of our models on validation data. AutoML's primary evaluation metric is AUC: area under the (receiving operating characteristic) curve.

AUC	Occurrences	Example classes
1.0	39	swing set, umbrella, wheel
0.99	191	shark, tooth, violin
0.98	78	fork, scorpion, trumpet
0.97	17	eraser, fire hydrant, pool
0.96	13	bottlecap, camouflage, diving board
0.95	1	cooler

Table 1. Performance of baseline binary classifiers on validation data.

Our average AUC across binary classifiers achieved 98.66! While our baseline models performed excellently on the validation data, we ran into several significant challenges extending our results to multiclass classification stemming from our use of Google Cloud and one-vs-rest (which ultimately prevented us from making a full Kaggle submission).

5. Data Augmentation using Generative Adversarial Network (GAN)

To improve the performance of classifiers trained using AutoML, we first tried to augment the dataset using Generative Adversarial Network. A generative adversarial network (GAN)[2] consists of two contesting neural networks: a generator and a discriminator. The generators goal is to synthesize realistic test data and the discriminators goal is to differentiate synthesized examples from real ones. We experimented with augmenting our dataset using two GAN variants: unconditional and deep convolutional (DCGAN)[1, 3].

In our study, we found that unconditional GAN suffered from vanishing or exploding gradients from discriminator to generator[6]. The generator produced unrealistic garbage images that the discriminator was unable to recognize. DCGAN was more successful in our tests, so we focused our work on optimizing and scaling DCGAN.

5.1. DCGAN Architecture

Generator:

- An Embedding Layer with a $7 \times 7 \times 64$-dimension output.
- A 2D Convolutional Layer with 64-output filters and a 5-by-5 kernel window, the strides of the convolution along the height and width is $(1, 1)$.
- A 2D Convolutional Layer with 32-output filters and a 5-by-5 kernel window, the strides of the convolution along the height and width is $(2, 2)$.
- All Embedding Layer and Convolutional Layers are followed by a Batchnormalization Layer.
- Generator hidden layers use ReLU activation with a tanh activation output.

Discriminator:

- A 2D Convolutional Layer with 64-output filters and a 5-by-5 kernel window, the strides of the convolution along the height and width is $(2, 2)$.
- A 2D Convolutional Layer with 128-output filters and a 5-by-5 kernel window, the strides of the convolution along the height and width is $(2, 2)$.
- A Flatten Layer.
- An output Dense Layer with a 1-dimension output.
- All Convolutional Layers are followed by a Dropout Layer with a 0.3 rate of the inputs units to drop.
- Discriminator hidden layers uses LeakyReLU activation.

5.2. Data Augmentation using DCGAN

Datasets related to the twelve worst performing baseline classifiers (Table 2) were augmented using DCGAN in our study. As an example, synthetic bottle caps generated by DCGAN along with their nearest neighbours in the training dataset are shown in Figure 3.

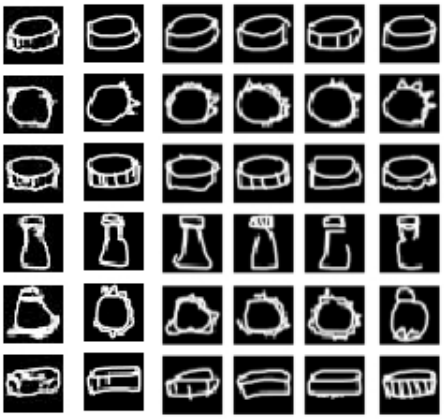

Figure 3. Synthetic bottle caps sketches generated using DCGAN (left column); Nearest neibours found from the training data, based on L2-norm distance (next five columnes).

5.3. Practical Implementation of DCGAN

We initially trained our best-performing DCGAN on a CPU locally. Each epoch took 10 minutes, so training ran for nearly three days. In order to utilize all of the available data (not just 10k examples in initial tests) and build models for all worst performing classes, we implemented our code on an available cluster and leveraged Google's cloud GPU instances. The improved implementation speed up each epoch to around 20 seconds, and augmentation of each class only took about 2 to 3 hours now.

	AUC	Accuracy at 0.5 threshold
diving board	0.96	88.4
garden hose	0.96	89.1
frog	0.96	89.2
bottle cap	0.96	89.5
mouse	0.96	89.5
pliers	0.96	90
lobster	0.96	90.1
raccoon	0.96	90.1
trombone	0.96	90.1
camouflage	0.96	90.2
string bean	0.96	90.2
marker	0.96	90.9

Table 2. Worst performing classes.

6. Data Augmentation using Transformation and Stroke Removal

We attacked the problem of data augmentation from another angle with image-based transformations. Instead of training a neural network to generate images drawn from the same underlying distribution, this work focused on increasing the amount of training data through techniques like scaling, translation, and rotation as well as stroke removal approach.

Based on the baseline model (AutoML) training, we selected the diving board image class to test the performance impact of different transformations. The baseline diving board classifier's precision ratio was only 88.391%: one of the lowest among the baseline classifiers. We investigated the misclassified validation examples to gain insight into the baseline model's challenges. We surmised that we could generate more training examples like the challenging ones depicted below with geometric transformation, morphological transformations and stroke removal respectively.

Figure 4. False Negatives

6.1. Geometric Transformation

We applied various 2D geometric transformations using OpenCV. The transformations preserve the image content but deform the pixel grid and map this deformed grid to the destination image. Below we depict three original airplanes from the original dataset and describe a variety of transformations we have explored.

- **Scaling**: simple image resizing. When applying the resize transformation, we used a scaling factor of 0.5 for height and width. OpenCV offers various interpolation methods, and we picked **cv2.INTER_CUBIC** as the interpolation method for zooming.

- **Rotation**: calculates an affine matrix of 2D rotation. We experimented with 45 degree rotations.

Figure 5. Rotation Examples

- **Translation**: Translation is simply the shifting of objects location.

6.2. Morphological Transformations

Morphological transformations are simple operations based on the image shape. They are typically performed

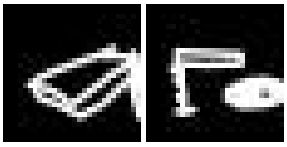

Figure 6. Translation Examples

on binary images. A morphological operation needs two inputs: an original image and a structuring element or kernel which decides the nature of operation. In our tests we experimented with the kernel all-ones matrix[10]. Two basic morphological operators are erosion and dilation.

- **Erosion**: the kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel are 1, otherwise it is eroded (zeroed out).
- **Morphological Gradient**: This is the difference between dilation and erosion of an image. The result will look like the outline of the object.

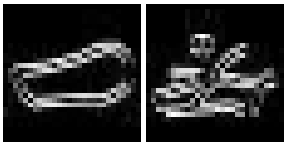

Figure 7. Morphological Transformations Examples

Next we plan to explore other packages like Augmentor and stroke-based transformation methods such as stroke removal and stroke deformation[8]. We will test the efficacy of these data augmentation efforts by applying them to our baseline model.

6.3. Stroke Removal

Inspired by the Sketch-a-Net multi-channel model, and considering that we had access to the underlying stroke data, we produced additional training images from the vector data via stroke removal. The original quickdraw sketches represent a sketch as list of strokes with corresponding timestamps. By utilizing the different timestamps, each stroke can be separated out with a [X,Y,T] format where X, Y is 256 array and T represents time. To match our other training images, we transformed each (X,Y) combination into 28×28 grayscale bitmaps where one stroke was omitted. Below we show the different images generated via stroke removal on an example diving board sketch.

Stroke removal was particularly applicable in the context of our problem since the sketches were collected from a time-limited game. We observed many partially-drawn sketches in the dataset, and stroke removal allowed us to give these incomplete sketches better representation in our training data. This was important since our baseline model was in some cases unable to correctly identify partial sketches.

Figure 8. Stroke Removal Examples

6.4. Training in AutoML with transformed images

We tested our image-based transformations by running augmented training sets through our binary diving board classifier and observing the performance impact. The original dataset contained 290,238 diving board sketches. We applied the below transformations on each of them to augment the base set of diving boards.

- For geometric transformation, we applied rotation and translation.
- For morphological transformation, we applied morphological gradient as it added the most diversity of the operations we tested.
- For stroke removal, we focused on removing the final stroke to simulate timing out before completing a sketch as closely as possible.
- We trained a final model which combined all of these transformations.

At the time of this paper's writing, we're still collecting results from training these models with augmented training data. We expect to have results that we can share for the poster session.

7. One-vs-rest Challenges

There were three issues which compounded on one another to make multiclass classification difficult. Firstly, the number of model queries necessary to make a single image classification was equal to the number of classes (340 in our case). In order to process our entire test set (112,199 images), we needed to make 340 queries / image $\times$ 112,199 images = 38,147,660 queries! While we were collecting our results, Google Cloud did not support making offline or batch predictions. To make matters worse, they enforce strict rate limiting caps which kept us from running through all of the test examples. Ultimately, we were only able to make 55% of the 38 million predictions which led to a Kaggle score of 0.4.

Another aspect of one-vs-rest is the need to break ties when multiple models make a positive prediction. AutoML offers confidence scores between 0 and 1 for each classification which allowed us to simply choose the highest confidence score in multiclass prediction. However, the score scales needed to be reconciled between classifiers. A well-known approach to this problem which we pursued is Platt scaling[11] in which logistic regression classifies individual model scores into class probabilities.

The last issue was the difference in class distribution between the training datasets we prepared for each classifier (where 50% of the images were positives) and the multiclass test dataset (where an average of $1/30 = 0.2\%$ were positives). Its important in multiclass classification (and machine learning in general) that your training data has similar consistency to test data, and ours was not.

In hindsight, we would have made a few changes to produce better experimental results. AutoML was unable to match our needs. We chose to attempt one-vs-rest multiclass classification because AutoML was only able to support one million images per model, but it proved to be a difficult strategy to execute well given the aforementioned prediction count, tiebreak, and class distribution issues.

8. Code Submission

All our codes have been uploaded to Github. Since GCP's services are paid, unauthenticated predictions cannot be made with the models we trained in our research.

Link:https://github.com/lambdabaa/quickdraw

References

[1] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan. *Gan-based Synthetic Medical Image Augmentation for Increased CNN Performance in Liver Lesion Classification*. arXiv preprint arXiv:1803.01229, 2018.

[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. *Generative Adversarial Nets*. In Proceedings of NIPS, pages 2672 2680, 2014.

[3] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. *Improved training of Wasserstein GANs*. CoRR, abs/1704.00028, 2017.

[4] Link: Using Machine Learning to Explore Neural Network Architecture

[5] A. Mikoajczyk, M. Grochowski. *Data Augmentation for Improving Deep Learning in Image Classification Problem*. 2018 International Interdisciplinary PhD Workshop (IIPhDW), DOI: 10.1109/IIPHDW.2018.8388338, 2018.

[6] L. de Oliveira. *Introduction to Generative Adversarial Networks*. [Online].

[7] J. Wang and L. Perez. *Ramer-Douglas-Peucker algorithm*. Wikipedia.

[8] Q. Yu, Y. Yang, Y. Z. Song, T. Xiang, and T. Hospedales. *Sketch-a-net that beats humans*. British Machine Vision Conference (BMVC), 2015.

[9] B. Zoph, Q. V. Le. *Neural Architecture Search with Reinforcement Learning*. ICLR, 2017.

[10] Link: Morphological Transformations

[11] Link: Platt Scaling