Machine Learning Engineer Nanodegree

Capstone Proposal

Gopala krishna kumar Thangavel

February 27th, 2019

Proposal

Project – **Quick, Draw! Doodle Recognition Challenge** (from Kaggle)

Domain Background

I have taken this problem statement from Kaggle. This problem statement comes from the domain of Gaming done using Image classification. "Quick, Draw!" was released as an experimental game to educate the public in a playful way about how AI works. The game prompts users to draw an image depicting a certain category, such as "banana," "table," etc. The game generated more than 1B drawings, of which a subset was publicly released as the basis for this competition's training set. That subset contains 50M drawings encompassing 340 label categories. Since the training data comes from the game itself, drawings can be incomplete or may not match the label. The 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. By advancing models on this dataset, pattern recognition solutions can be improved broadly and they can be used for scenarios like Handwriting recognition, OCR (Optical Character Reading), ASR (Automatic Speech Recognition) etc.

Problem Statement

Even though this use case comes from a gamin background, I'm planning to take up the part of simple classification of pre-made custom doodles of any objects/things (one among the 340 categories). So in essence, the problems statement is to categorize doodles drawn by user that fits in to the category set of 340 labels. This can be achieved using neural networks based on image classification.

Datasets and Inputs

The Quick Draw Dataset is a collection of millions of drawings across 300+ categories, contributed by players of Quick, Draw! The drawings were captured as timestamped vectors,

tagged with metadata including what the player was asked to draw and in which country the player was located.

Two versions of the data are given. The raw data is the exact input recorded from the user drawing, while the simplified version removes unnecessary points from the vector information. (For example, a straight line may have been recorded with 8 points, but since you only need 2 points to uniquely identify a line, 6 points can be dropped.) The simplified files are much smaller and provide effectively the same information.

For this problem, either datasets can be used to solved or both can be used together. I'm planning to use simplified files for starting phase, since it will give an edge in terms of time and computation load.

Regarding the distribution of target classes, the count of datasets belonging to certain categories vary from 100,000+ to 300,000+. Whereas most of categories fall under 100,000 to 200,000 (almost around 80%) rest 20% are distributed till ~340,000 nearly. So there are enough data points to train each and every category so we can assume that the data points are well balanced.

All doodles are provided in vector (matrix) format. It varies on number of strokes and color used by user for each stroke. So the data needs to preprocessed and restructured a bit so that all data will be of similar shape before being fed in to the network layers.

Solution Statement

This problem can be solved using Neural networks that can be used for image classification. The solution can be build up on any of the pre-trained models like ResNet-50 (trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks). Over that pre-trained model, custom layers can be added which are specific to the current scenario so that the output corresponds to the 340 labels of categories appropriate for the problem.

Benchmark Model

I'm planning to consider the model provided by one of the competitors in Kaggle platform for the same problem statement, as my benchmark model. Also this model can be verified in the following link: https://www.kaggle.com/nicstar92/quick-draw-doodle-recognition-challenge . From the code part given we can get to know that overall final score of this benchmark model is around ~0.65 (f1-score), and hence this model can be used as benchmark.

Evaluation Metrics

I'm planning to majorly use following 2 evaluation metrics: 1. Accuracy score and 2. F1-score. Since the problem statement is a classification based scenario and it has quantifiable categories with datasets that are well balanced, both accuracy score and f-1 score will be best options to

evaluate the model's performance. Adding to that using F1-score also will provide an advantage while reducing the overall loss value since it considers precision and recall values. Also F1-score will be helpful for evaluating the model against the benchmark model (contender have used f1-score to evaluate the model).

Project Design

First step is to load the data and have to generate some charts that clearly shows how the data has been distributed. So that we can understand some statistics of data (for example, total count of data, total count of distinct labels, total count of records available for some random labels etc). Next step is to do cleanup of data if at all it is necessary and also preprocessing. Basically restructuring data as per our convenience and use case. So that it will fit well with the model we create.

Next step is to import any of the pre-trained models like ResNet-50 that has been already trained with many sample images from ImageNet. Then that base model has to be trained with the training data and has to be checked how well it performs with the testing data. Now we will have overall rough idea in terms of data and our base model's level.

Next step is to multiple required neural network layers over the base model in such a way that model suits more aligned with the type of data and use case. Once we make the model better than base version, it has to be trained, tested and validated against training, testing and validation data. Based on the performance (scores like accuracy, f1) of the model, we have to tune the hyper parameters of the model (for ex: number of nodes, value for weights and biases etc.). The above process of training, testing, validating and performance tuning has to be repeated until we reach the desired score values and that should be it.