GAN generated and Hand-Drawn Scribble Identification

Shubham Malik
Department of Computing Science
Simon Fraser University
shubham_malik@sfu.ca

Akash Singh Kunwar
Department of Computing Science
Simon Fraser University
akash_kunwar@sfu.ca

Aditi Shrivastava
Department of Computing Science
Simon Fraser University
aditi_shrivastava@sfu.ca

Rohan Harode
Department of Computing Science
Simon Fraser University
rohan_harode@sfu.ca

Abstract

This paper explores the use of Generative Adversarial Network (GANs) to produce scribbles of different categories. In the project, we also built a multi-class classifier to assign GAN generated or Hand-Drawn scribbles into 7 unique categories. The GAN model generates scribbles with minimal generative loss of 3% and the classifier guesses the correct category with an accuracy of 87% and 93%, implemented on multi-layer perceptron and convolution neural network, respectively, whilst also providing a comparison among other neural network classifiers like random forest and K-nearest neighbors. Experiments were carried out on images from Google's quick draw [1], which is an open-source dataset of 50 million drawings, and outputs from different models were observed.

1 Introduction

GANs were introduced by Ian Goodfellow et. al. as "a new framework for estimating generative models via an adversarial process" in a 2014 paper [2]. The network comprises of a generative model that is trained to generate the data to trick the discriminative model, whose job is to classify between 'real training data' and 'fake generated data'. By forcing the network to compete, the generative model is directed to improve its output until the fake generated data is indistinguishable from real training data [2].

In 2016, Google released "Quick Draw", an experimental game launched to educate people about how AI works which challenges people to draw a given object in 20 seconds. By advancing models on the dataset, we can improve pattern recognition solutions more broadly, which has immediate implications on hand-writing recognition and its application in the areas including Optical Character Recognition (OCR) and Natural Language Processing (NLP). In this paper, we will explore the use of GAN to generate scribble and then passing the GAN generated image or hand-drawn scribbles to the classifier to categorize it among 7 different categories. The challenge comes with making the model generate highly accurate scribbles, which requires extensive training while minimizing the loss and then training a classifier that predicts the categories accurately while being fast to work in real time.

Similar to this work, Walmart engineer Kirti Pande and Vincent Kuo used the Quick draw dataset [3] to architect a convolutional neural network to tackle sketch classification which predicts the scribbles with an accuracy of 87% on KNN model. We borrowed the idea of comparing our model with different classifiers like MLP and random forest, but we do not extensively compare with all classifiers due to the limit of time.

2 Approach

We began to understand the structure of the array that makes up the image. Then we dig deeper to fit some basic classifiers and basic neural network. From there we identified different classifiers we can for our predictions. To determine the success of our classifier's prediction and sketch accuracy of our GAN model, we have used multiple classifiers to predict our scribble(hand-drawn on canvas or gan generated) category. The goal was to classify each scribble with maximum accuracy and avoid miss-classification and efficiently train the GAN model such that it gives minimum loss(GAN loss in section 3.5.2).

2.1 Dataset

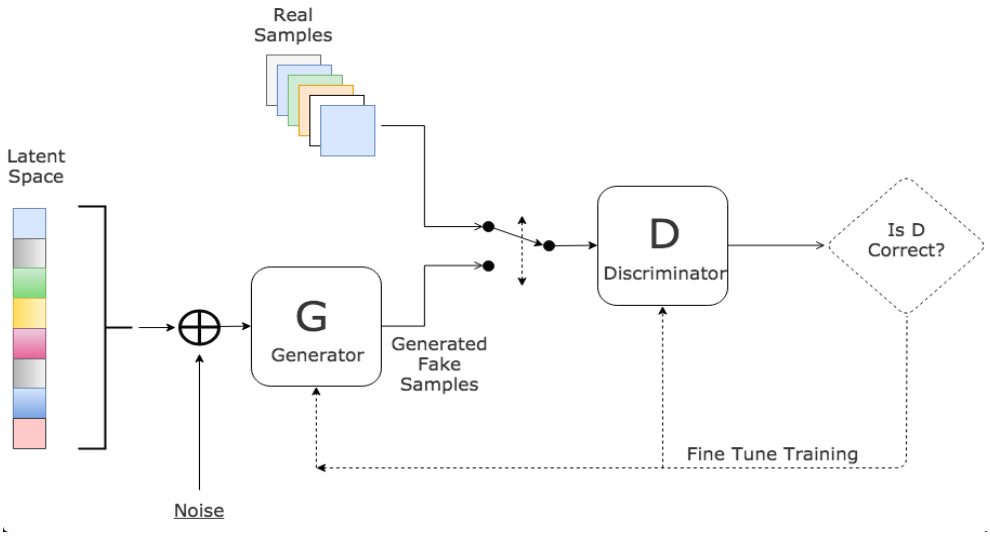
The dataset, we are using is Quick Draw dataset released by Google[1]. It consist of 28 x 28 grayscale bitmap and converted to a 784 dimensional vector. For GAN model, we are using 75% of the total number of images of one class to train the GAN model. To build our classifier, we are using a total of 7 categories, each having 2000 images. We defined our training set to be 80 percent, validation set to 15 percent and test set to 5 percent. Resultantly, we get 11200 images for training set, 2100 images for validation set and 700 images for the test set. The same number of images of every class goes to training and test set, making each category proportionate.

Sample images of different categories

2.2 Training Classifiers

To get the best model after training, we have used two callback functions: early stopping and model checkpoint. With early stopping callback function, we are monitoring the validation loss and it stops the training once the validation accuracy starts to decrease, whereas, model checkpoint function is used to save the best model. We have added a patience parameter to run for a few more epochs after the validation accuracy decreases. We will pass these callback functions to keras fit function to save the best model, together these callback functions give us the best performing model in a shorter time and thus saving computational resources.

2.3 Training on GAN



Our model is vanilla GAN with a discriminator and a generator. Generator network creates fake image from noise and tries to fool the discriminator by classifying its output as real. Discriminator

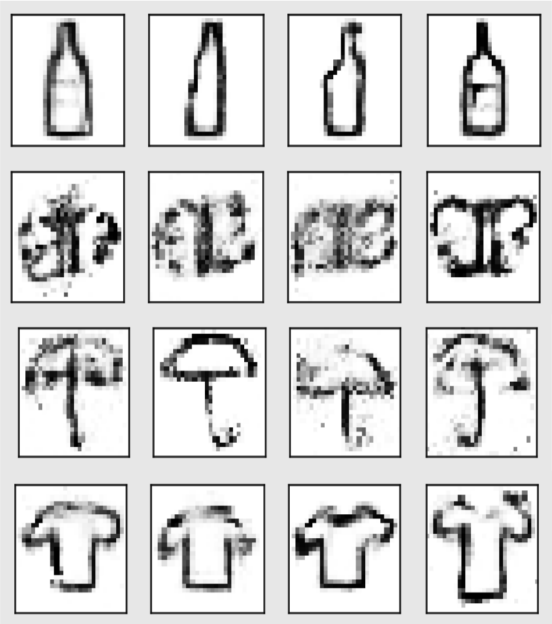
network compares real input images with generated (fake) images. The output of the discriminator network penalizes generator with generator loss so that it can modify its generated image to make it more realistic.

Initially, we used Batch normalization in the generator network for producing a higher quality of images. However, we didn't notice any significant effect as it was generating the same quality images for the same batch. We then moved to the gradient penalty method using Adam Optimizers to reduce generator and discriminator losses.

Generator: Inputs are a 100-dimensional Gaussian noise vector *z_size*. Outputs a 28x28x1 (*g_out*) which is bounded between +1 to -1 since the final layer has *tanh* activation function. No batch normalization is used in hidden layers.

Discriminator: Input to discriminator is 28x28 generated image and 28x28x1 real image. The output is a single logit along with final output of network i.e. d_out (real/fake), logits (real/fake) corresponding to real or generated image. The activation function of Discriminator's final layer is *sigmoid*, making its output restrict between (0,1). Batch normalization is not used after the activation of convolutional layers.

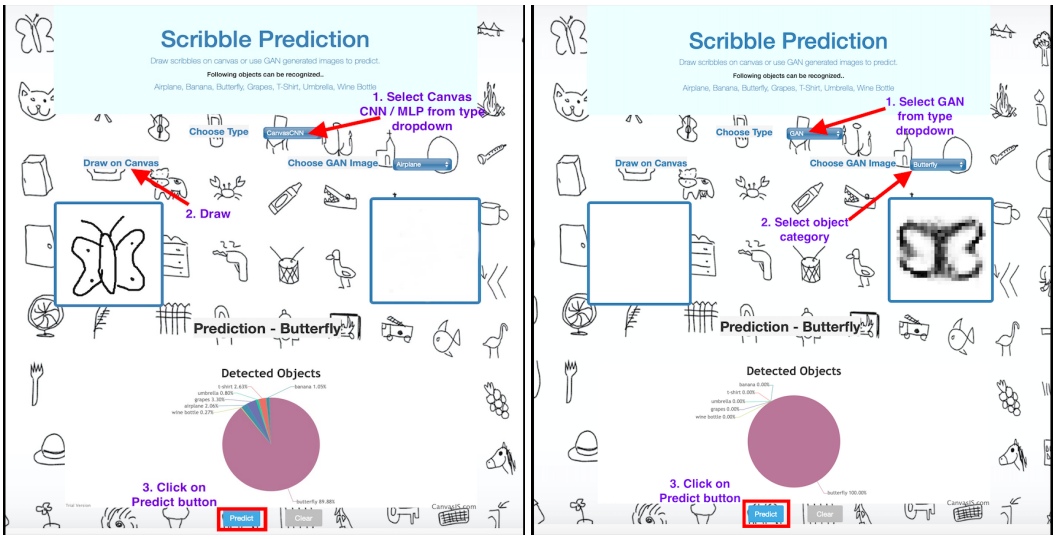
Hidden Layers: We used *Leaky ReLU* as activation for hidden layers in the network as they helped the model to perform better and few neurons would die while training once they passes through the Adam optimizer. The figure below shows the images generated by the GAN model with 3% maximum loss.



GAN generated samples

2.4 Webapp

We have created a web application where we put our trained models to test. The application allows users to make predictions of our GAN generated images or their own hand-drawn scribbles. Firstly, the user can select either to draw on Canvas or choose any one of GAN generated images of 7 trained objects. To draw on Canvas, users can select either CNN or MLP model to make predictions whereas for GAN images we have selected CNN as the default model for prediction. The predict button passes our image data bytes and our dropdown selected values to the flask controller. After preprocessing the image, model's predict method returns the predicted percentage of all the classes and we can see it in the webapp in the form of a pie chart. Figure 1 below shows the prediction using CNN model and hand-drawn image of butterfly, second figure shows the prediction on GAN generated image of butterfly.



3 Experiments

3.1 Random Forest

The first machine learning technique we used was Random Forest(RF). Considered as the most flexible classification algorithm, Random forest creates decision trees on randomly selected data, gets the prediction from each tree and selects the best solution by the means of voting. We used the standard RandomForestClassifier library from scikit-learn and utilized GridSearchCV to cross-validate the model and optimize parameters. We experimented with the number of trees in our random forest and finally settled with n_estimators = 100 as our final model.

3.2 K Nearest Neighbor

We then tried the k-Nearest Neighbor (KNN) machine learning algorithm on our dataset. The algorithm assigns an unknown data point to a class by finding the most common class among the k-closest examples. We used the standard KNeighborsClassifier library from scikit-learn and similar to Random Forest, used the GridSearchCV to cross-validate the model and optimize parameters[4]. Our final model used k = 5.

3.3 Multilayer Perceptron

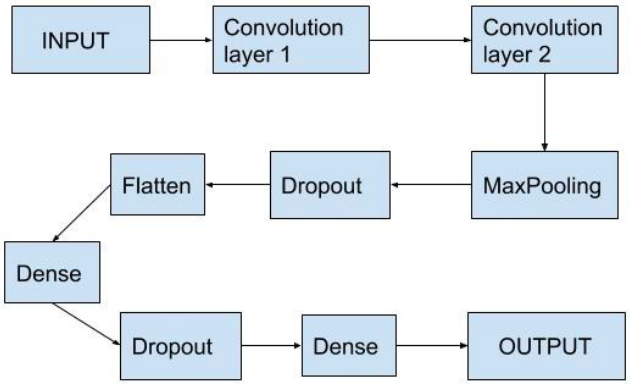
Before using the model, we have loaded the dataset with reshape value set to false because the images are needed to be flattened in the case of MLP. We are using model sequential API to create models layer-by-layer. Our input layer dimension is 784 (28 x 28). We are using the activation function *ReLU*. We have also added a Dropout to avoid overfitting. For the final output layer, we have a dimension equal to the number of classes we used in the training. The output layer has a *softmax* activation function. The disadvantage of using MLP is that the parameters can grow very high and also neglects spatial information as its inputs are flattened vectors. This motivated us to shift to CNN model.

3.4 CNN - Convolution Neural Network

Convolutional Neural Network model is well known for analyzing visual imagery, the most popular architecture for image classification[6]. Data preprocessing involved normalizing the values for each image sample. This ensures that each feature has a similar range and the gradient doesn't become uncontrollable during backpropagation. We then added labels to our images followed by one-hot encoding for the labels.

The model architecture is shown in the figure below. For a scribble of size 28 x 28 x 1, we first run the image through two convolutional layers with 32 and 64 filters respectively and a kernel size of 3 x 3, with stride one and *ReLU* activation function. We also add a zero-padding border around the image so that the resulting outputs have the same dimension. The output then goes through a max-pooling layer with a kernel size of 2 x 2. Following this, is a Dropout layer with a rate of 0.25, which helps prevent overfitting. Finally, the tensor was flattened and passed through a fully connected dense

layer with the *ReLU* activation function along with a Dropout layer. The output then goes through one last transformation to produce logits of dimension 7 (total categories) with *softmax* activation that assigns probabilities to each class. The model was trained using cross-entropy loss and Adam optimizer.



3.5 GAN Experiments

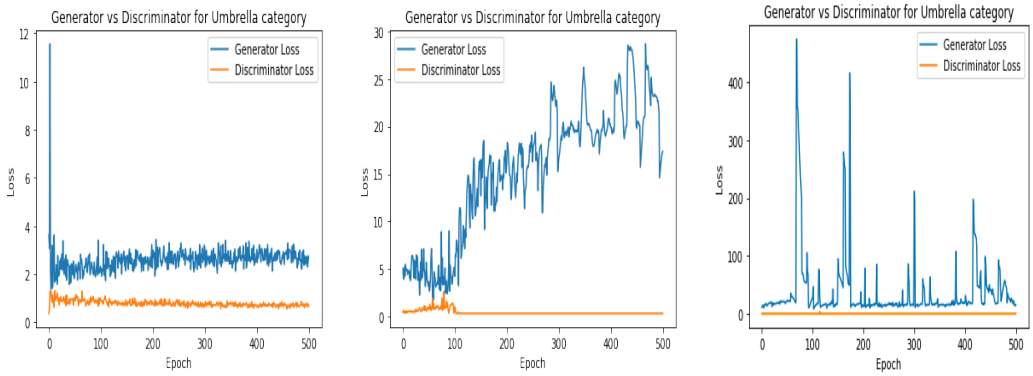
3.5.1 Adjusting Hyperparameters $(learning\ rate, \beta_1, \beta_2)$ for GAN

The default learning rate for generator and discriminator in the vanilla GAN is 0.002. While training the model, we found the generator loss decreased slowly and converges to a stable value of 2.8 after a considerable number of epochs. The discriminator loss is always low around (0.7 - 0.9). To increase the accuracy of our model and train within an acceptable amount of duration, we decided to decrease the generator and discriminator learning rate to 0.0001. This decreases generator loss to a great extent, and we could record loss values in a range of 1.7 to 2.1.

Default hyperparameters values for both generator and discriminator are β_1 = 0, β_2 = 0.9[5]. But, after changing the values of hyperparameters to β_1 = 0.5, β_2 = 0.99 we observed a significant decrease in generator loss (2.2 reduced to 0.7)

3.5.2 Adjusting number of hidden layers of Generator and Discriminator

We assessed our model's accuracy to the number of hidden layers in both the generator and the discriminator network. We discovered that, increase in the number of hidden layers in our model (might be specific to our dataset) results in increasing the generator loss to higher values over epochs while the discriminator loss remains stable at around 0.32. The most optimal design was to include only 1 hidden layer with *ReLU* activation function and pass it to *tanh* and *sigmoid* activation functions for the final output of generator and discriminator respectively. Below are the loss graphs for model with various hidden layers:



Hidden Layers = 1 Hidden Layers = 2 Hidden Layers = 3

4 Conclusion and Future Work

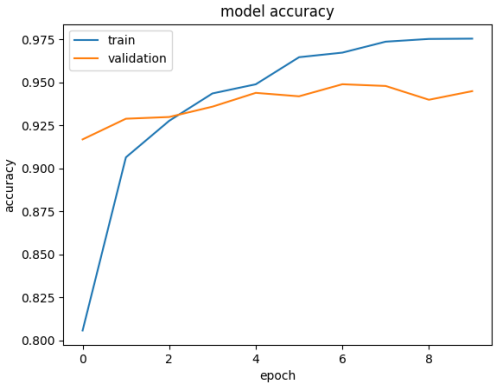
4.1 Conclusion

From the comparison among different classifiers in the table below, it is evident that the best results were obtained using Convolutional Neural Network (CNN)[4] followed by Multilayer Perceptron (MLP). The predicted probabilities for the correctly identified object were quite high in the case of CNN, whereas, in the case of MLP, the predicted probabilities were scattered and there wasn't any clear majority even for the correctly identified object in some cases. The correctness of images generated using GAN models was exemplary, the CNN classifier was able to predict GAN generated scribbles with more than 98% accuracy.

4.2 Future Work

We approached this problem by dividing our samples to 7 different categories, whereas in reality there are a lot many categories of which scribbles can be drawn. It is sometimes difficult to categorize indistinguishable object drawings, for instance an envelop and door, fence and spreadsheet. The work done by us could be extended in various directions, we will train our model on similar looking scribbles and overcome the model's inability to recognize indistinguishable features and subsequently predicting the sketch more accurately. Also, due to the high computational requirement of GAN, the model currently operates on 7 objects only. In the future, we plan to expand our code to identify a wider variety of objects. We also plan to implement more complex architectures like VGGNet and AlexNet to obtain a model with even higher accuracy.

Model	Parameter	Accuracy
Random Forest	100 trees Max depth = 8	0.79
KNN Classifier	N Neighbors = 5	0.83
MLP	Hidden layers=(,784)	0.87
CNN	No. of filters = 32 → 64	**0.93**



4.3 Contributions

Tasks	Members
Topic and Data Set Selection	Aditi, Akash, Rohan, Shubham
CNN, KNN implementation and research	Aditi, Akash
RF implementation and research	Aditi, Rohan
MLP implementation and research	Akash, Shubham
Training different classifiers	Aditi, Shubham
Testing and accuracy improvement	Akash, Rohan
GAN implementation and experiments	Rohan, Shubham
Webapp integration	Aditi, Akash, Rohan, Shubham
Poster	Aditi, Akash, Rohan, Shubham
Report (in Latex)	Aditi, Akash, Rohan, Shubham

References

[1] GoogleCreativeLab, The Quick, Draw! Dataset. https://github.com/googlecreativelab/quickdraw-datasetthe-raw-moderated-dataset, 2016

[2]Ian J. Goodfellow, Jean Pouget-Abadie Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio.:*Generative Adversarial nets*

[3] Akhilesh Reddy, Vincent Kuo, Kirti Pande, Tiffany Sung, and Helena Shi.:*Doodling with Deep Learning!*

[4] Phan Thanh Noi and Martin Kappas.: *Comparison of Random Forest, k-Nearest Neighbor, and Support Vector Machine Classifiers*

[5] Karras, T., Aila, T., Laine, S., Lehtinen, J.: *Progressive growing of GANs for improved quality, stability, and variation*. In: ICLR (2018)

[6] Juan Carlos Gonzalez: Use of Convolutional Neural Network for Image Classification

[7] Diego Gomez Mosquera, Google Engineer: *Gans: Generative Adversarial Networks*