Comparative Study of Image Classification Algorithms used in AI Doodling

CS7IS2 Project (2019/2020)

Anusha Lihala, Chavvi Chandani, Kanika Ghiloria, Sarvani Chakrabarty

lihalaa@tcd.ie, chandanc@tcd.ie, ghilorik@tcd.ie, chakrasa@tcd.ie

Abstract. This paper analyzes the performance of a diverse set of algorithms in predicting hand-drawn doodles using a subset of the dataset retrieved from Google's online game "Quick, Draw!". The experiments are carried out using Convolutional Neural Networks, K-Nearest Neighbors, Support Vector Machine, Logistic Regression, Multi-Layer Perceptron and Random Forest Classifier. A comparative study is, then, carried out amongst the algorithms based on time and code complexity and using Accuracy and Confusion Matrix as the evaluation metrics

Keywords: CNN, K-Nearest Neighbors, SVM, Random Forest Classifier, Logistic Regression, Multi-Layer Perceptron, VGG16, SqueezeNet, Accuracy, Confusion Matrix, Image Classification, Quick Draw, Doodles, Hand Drawn Images

1 Introduction

The pervasive nature of touch screen devices have made research on sketch-based image retrieval (SBIR) to become a popular research topic. SBIR generally takes a hand drawn sketch as input and gives a matching image from the data pool as output [9].

The research on SBIR took an exponential growth when a new game was released by Google in November 2016 called the "Quick! Draw". The game provided an extensive database of hand-drawn doodles which are stored by the stroke in the dataset and can be accessed by the crowd for various purposes. There are 345 categories of approximately 50 million drawings overall in the dataset. In order to address the challenges faced in computer vision and pattern recognition, in general, of managing noisy data which might have several representations under the same category, the Quick! Draw dataset seemed appropriate for use in this project. The paper attempts to perform image classification on the preprocessed version of the dataset, which renders the doodles into 28x28 grayscale bitmaps in numpy .npy format and aligns them "to the centre of the drawing's bounding box rather than the top-left corner" [2]. Furthermore, the following 15 categories are considered as opposed to the original 345: Apple, Bowtie, Candle, Door, Envelope, Fish, Guitar, Ice Cream, Lightning, Moon, Mountain, Star, Tent, Toothbrush, and Wristwatch.

Learning features from this dataset, we believe, will prove to be useful in understanding how adequately the chosen algorithms work in correctly correlating the doodles with their categories. Sub-fields of Computer vision such as Optical Character Recognition stand to benefit from such advent of robust classifier systems, especially for highly noisy data [1].

2 Related Work

Guo, James and Eric architected and implemented Convolutional Neural Network (CNN) and K-nearest neighbours (KNN) algorithms on the Quick, Draw! Online dataset [1], evaluating which they've determined discrete attributes of the dataset. Accuracies of 34.4% and 62.1% were achieved by KNN and CNN respectively. Mean Average Precision @3 (MAP@3) was used as the Evaluation Metric. Raw accuracy was too considered as an evaluation metric, but since the dataset consists of many categories, such as "cake" as well as "birthday cake", the research was taken ahead with MAP. As gathered from the accuracy scores, CNN outperformed KNN. The paper constructed saliency maps to analyze the reasons of KNN coming short of CNN despite correctly identifying objects within the same categories and concluded that KNN was able to only learn the general shape (for example the circle shape of an apple, blueberry and onion), whereas CNN correctly distinguished between blueberries and apples due to the presence of a stem. This paved the way to kickstart our project with these algorithms on a different sub-section of the same dataset. Also, as mentioned in the Future Scope of the paper, VGG is also experimented with on the dataset.

Ha and Eck in their paper [6] train a Recurrent Neural Network (RNN) on the Quick, Draw! Online dataset with the intention of studying sketch abstractions. However, their motivation was to recreate, expand or complete sketches from incomplete sketches as input rather than the classification of the images. Nevertheless, the paper gives insights into considering ordinal attributes in their RNN architecture which is lacking in our research.

Phu, Xiao and Muindi implemented and analyzed classical and deep learning models like Logistic Regression, Support Vector Machine, Convolution Neural Network and Transfer Learning to develop an efficient system to recognize labels of hand-drawn images from Google's QuickDraw dataset [3]. Logistic regression performed the best on 3 classes with an accuracy of 79.51% in 25s. On the other hand, it took 1089s with an accuracy of 43.89% to train 50 classes. It was noticed that Logistic Regressor misclassified bananas as hockey sticks. Apart from this, SVM was implemented with 4 different kernels, where RBF kernel performed the best with an accuracy of 62.68% for 3 classes in 317secs. Keeping this in mind we implemented SVM with RBF kernel too. CNN was experimented with 4 different variations, where the best performing CNN for 50 classes was V2 which attained a validation set accuracy of 81.83% depicting that the trained model is able to generalize well to unseen data. Transfer Learning has been proved to be very successful for artistic illustrations [4]. In this case it was implemented with 4 different architectures, namely VGG, Inception V3, ResNet50, MobileNet.

However, the best performing model on the QuickDraw dataset was also VGG (accuracy = 95.58, classes =3). It has the simplest architecture. This suggests that more complex architectures such as the "Inception Module" used in Inception V3 or the "Residual Block" used in ResNet50 may not be beneficial for the problem of doodle classification. As per Phu, Xiao and Muindi's research, simplified CNN performs the best in terms of both accuracy and training time.

Kim and Saverese used Caltech-4-Cropped dataset for image classification and compared the performance of SVM and KNN performance on airplanes, cars, faces, and motorbikes [5]. They found that SVM outperformed KNN classification, due to KNNs poor performance on car classification. If cars are not considered, it is noticed that SVM 's accuracy exceeded KNN only by 4%, otherwise there is a difference of 12% amongst the two.

Evangelyn D, Davu, Cynthia P and Jagannath D.J implemented and compared various machine learning classifiers on Google's Quick Draw dataset. Authors plan to feed the best performing algorithm into an algorithm designed for hearing and speech impaired people [7]. People can doodle their needs in the application and can be understood by other people. 10 out of 345 categories in the dataset are selected due to its large size. K-Nearest Neighbours, Decision Tree, Random Forest, Multilayer Perceptron, Support Vector Machine, Linear Support Vector Machine, and Gaussian Naïve Bayes Classifiers were chosen for the comparison and MLP gave the highest accuracy of 78%.

Bosch, Zisserman and Munoz compared the performance of Random Forest classifier to multiple kernel SVM classifier for the task of image classification by object category [8]. The model was trained in 30 images and tested on 50. It was concluded that M-SVM performs better with 81.3% accuracy while random forest gives the accuracy of 80.0%. Although M-SVM performs better, it was found to be 40 times more computationally expensive than random forest.

3 Problem Definition and Algorithm

Problem Statement: To understand how different algorithms perform in predicting hand-drawn doodles which have been retrieved from Quick, Draw! online dataset. Pipeline of the project is give below:

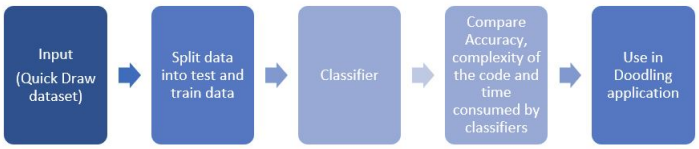

Fig. 1. Pipeline of the project

The codes were run on Google Colab with 77.90 GB of Disk Space using the Tensor Processing Unit. The following algorithms were then applied to the preprocessed dataset.

3.1 Support Vector Machine

Support Vector Machine is a supervised learning model for classification and regression problems. The algorithm creates a hyperplane which separates the data into classes. SVM looks for points (support vectors) closest to the hyperplane from both the classes. This distance between the line and the support vector is called the margin. Our goal is to maximize the margin. Maximizing the margin provides some reinforcement so that future data points can be classified accurately. Support Vector Classifier was implemented using Python's SciKit Learn framework.

from sklearn.svm import SVC

To test the model, we split the data into 2 parts : 90% for training and 10% for testing.

x_train, x_test, y_train, y_test = train_test_split(features, labels, random_state=0, test_size=0.1)

As per the knowledge gained after reading various papers on Image Classification, two kernels seemed to perform exceptionally good. Hence, SVM was implemented with two different kernels, namely 'linear' and 'rbf' to find empirically the kernel most suited for the task of doodle classification.

3.2 K-Nearest Neighbors

"Birds of the same feather flock together" is the concept behind K-Nearest Neighbors. The algorithm assumes that things which are similar to each other fall within close proximity. From background research, the algorithm has proved to be quite efficient. Hence, we implemented the K-Nearest Neighbors to the mentioned dataset. The basic flow of the code was:

- Import libraries
- Load dataset and convert them to array
- Split dataset into training and test datasets using *train_test_split(X, y, test_size=0.1, random_state=0)*
- Specify parameters of K-Nearest Neighbors using *KNeighborsClassifier(n_neighbors = 5, n_jobs=-1)*
- Predict results and evaluate using accuracy as the metric

3.3 Convolutional Neural Network using SGD and Adam optimizers

In the papers read regarding image classification, CNN proved to be performing very well constantly. CNN is a state-of-the-art model known for being able to recognize and quickly learn local features within an image [1].

A CNN architecture is developed using:

- Two convolution layers (5x5) which learn 32 and 64 filters respectively
- ReLU as activation function
- Two MaxPooling layers to reduce spatial dimensions:
  model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
- Two Dense layers of 512 and 218:
  model.add(Dense(512, activation='relu'))
  model.add(Dense(128, activation='relu'))
- Adam and SGD Optimizers
  - Choosing an optimizer is very important to the performance process of the algorithm. Research in related topics showed Adam and SGD optimizers to be the most extensively used.
  - Hence, we experimented using both the optimizers Cross-Entropy Loss function with their default respective learning rates.

The model is trained for 3 epochs each for both the optimizers used.

3.4 Logistic Regression

Logistic regression is a statistical method for binary classification. In order to perform multiclass classification either One-vs-All scheme or changing loss function to cross-entropy loss can be used. Cross entropy function minimizes the distance between two probability distributions. It uses optimal distribution, thus leading to better results. For instance, if a person draws a car, but instead of predicting it as a car the algorithm will predict it as a horse rather than a snake, because a car looks close to a horse when compared to a snake. Hence, we go ahead with Cross-entropy Loss.

Logistic Regression was implemented using Python's SciKit Learn framework.

from sklearn.linear_model import LogisticRegression

To test the model, we split the data into 2 parts : 90 % for training and 10 % for testing.

x_train, x_test, y_train, y_test = train_test_split (features, labels, random_state=0, test_size=0.1)

In order to use the cross entropy loss function, multiclass setting needs to be multinomial which is supported by 'lbfgs' solver. Maximum number of iterations taken for the solvers to converge was kept as default, i.e 100.

clf = LogisticRegression(solver = 'lbfgs', multi_class= 'multinomial', max_iter=100)

3.5 Random Forest CLassifier

Random forest is a multi-way classifier with a number of trees and each tree grows randomly [8]. Each node has the estimate of the image classes. The distribution across each node is aggregated to attain the class prediction.

Random Forest Classifier was implemented using Python's SciKit Learn framework.

from sklearn.ensemble import RandomForestClassifier

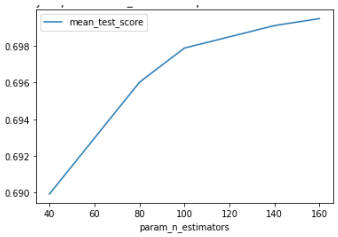

Fig. 2. Accuracy vs n_estimator plot for Random Forest Classifier

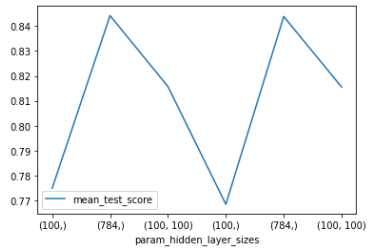

Fig. 3. Accuracy vs hidden_layer_sizes and alpha plot for MLP Classifier

GridSearchCV was used to cross-validate the model and optimize the parameters. It was observed that the curve is flattening after 100 trees (figure below). Hence, the number of forests chosen for the task were 100 with the rest of the default parameters.

parameters = 'n_estimators': [40,80,100,140,160]

classifier= RandomForestClassifier(n_jobs=-1, random_state=0)

rf = GridSearchCV(classifier, parameters, n_jobs=-1)

3.6 MLP Classifier

MLP Classifier uses neural networks for the task of classification. Random Forest Classifier was implemented using Python's SciKit Learn framework. We trained the data on the hidden layer size of (784,) and learning rate alpha = 0.001 for the best performance. The algorithm gave an accuracy of 84.68 % in 59.53 minutes.

from sklearn.neural_network import MLPClassifier

*tuning_parameters = 'hidden_layer_sizes' : [(100,), (784,), (100,100)],'alpha' : list(10.0 ** -np.arange(3, 5))*

clf_mlp = MLPClassifier(random_state=0)

mlp = GridSearchCV(clf_mlp, param_grid=tuning_parameters, n_jobs=-1)

3.7 Fine-Tuning Pretrained Models

We tried fine-tuning models pretrained on the ImageNet dataset, namely the SqueezeNet and VGG16 pretrained models available through PyTorch

[https://github.com/anushalihala/image-classifier].

For the fully connected layer, configurations of up to 3 layers with a number of hidden units ranging from 16 to 512 were tried. In the first set of experiments the pre-trained model features were fine tuned during training whereas in the second set of experiments the pre-trained model features were frozen and only the Fully Connected classifier was trained. The model failed to converge, despite tuning the learning rate, and seemed to keep getting stuck in local optima.

The model took roughly 2 minutes per mini batch of 32 iterations and was trained using CPUs only.

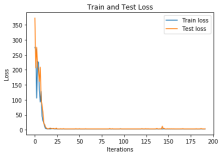

Fig. 4. Loss

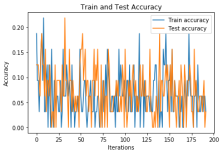

Fig. 5. Accuracy

4 Experimental Results

Logistic Regression, being the simplest of all the algorithms, is used as the baseline to compare the other algorithms with.

4.1 Evaluation Criteria and results:

Confusion Matrices Confusion Matrix for the two best performing algorithms along with the baseline algorithm have been plotted (Figures 6, 7 and 8) to visualize and compare the correct number of predictions for each category of doodles.

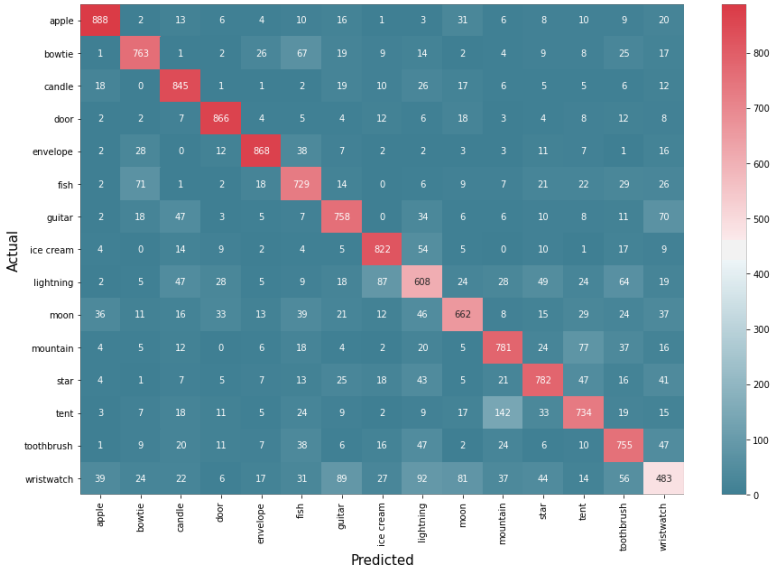

Fig. 6. Confusion Matrix of Logistic Regression

Accuracy Figure 9 illustrates the various algorithms we experimented with along with their accuracy in percentage(%) and the time taken in minutes. It

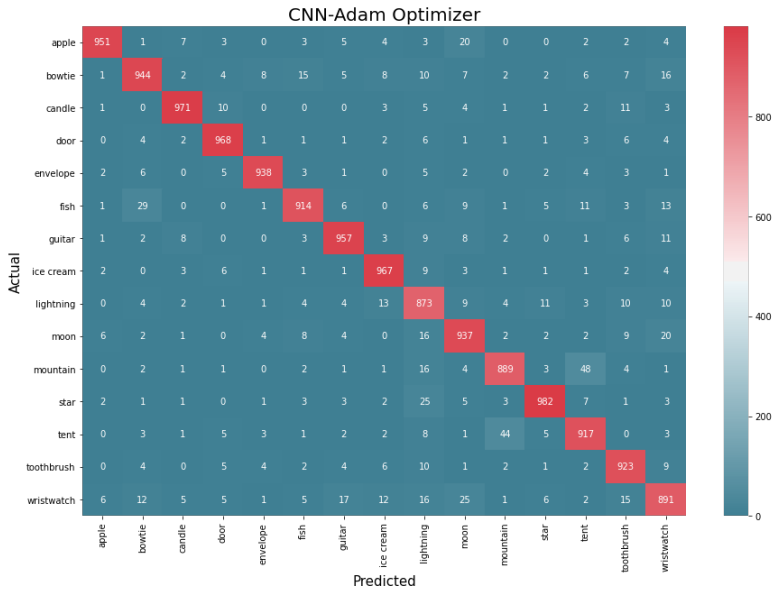

Fig. 7. Confusion Matrix of CNN - Adam Optimizer

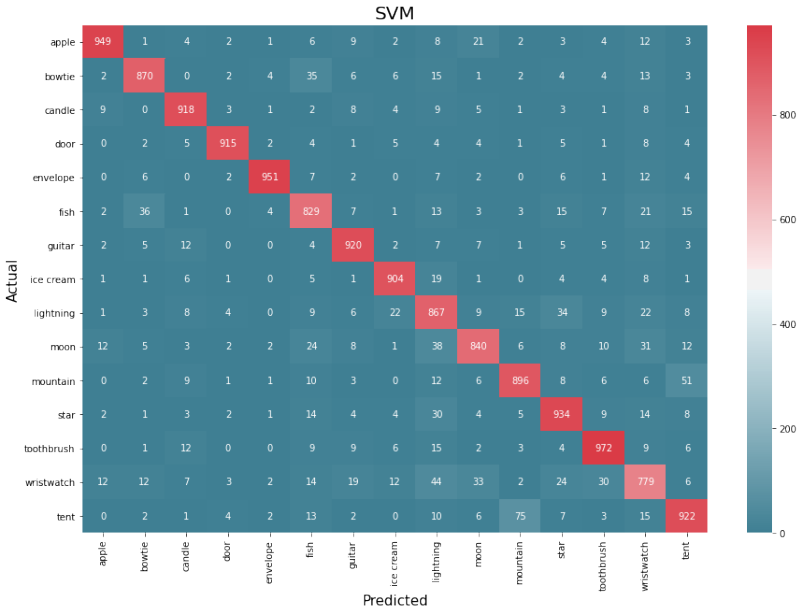

Fig. 8. Confusion Matrix of Support Vector Machine

clearly signifies that CNN using Adam Optimizer (93.4%) followed by SVM with 'rbf' kernel (89%) provide the highest accuracy. Thus, CNN proves to be the best for balancing both accuracy and training time.

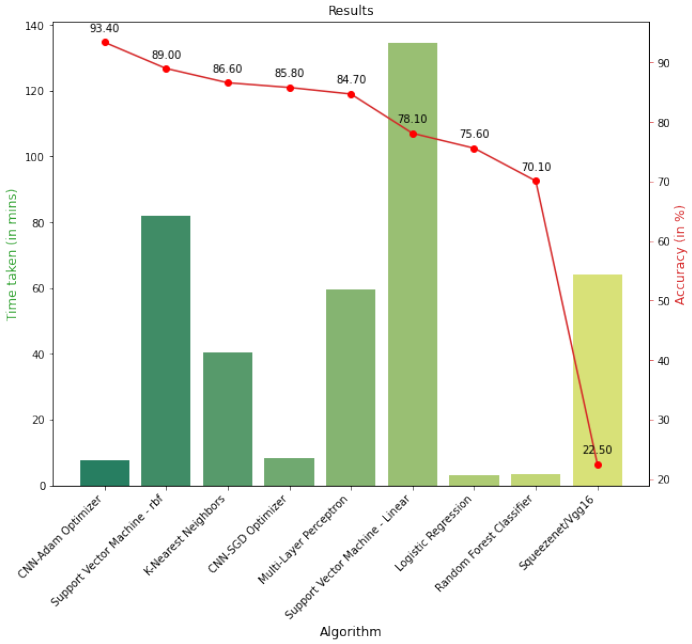

Fig. 9. Time and Accuracy plot for all Algorithms

4.2 Interpretation of Results

1. Few classification errors as visualized from the Confusion Matrices were:
   - A tent was classified as a mountain.
   - Lightning was classified as a star.
   - Bowtie was classified as fish.
2. As can be seen from Figure 9, 6 algorithms performed better than the baseline in terms of accuracy.
3. CNN using Adam Optimizer over 3 epochs gives the best accuracy among all the algorithms.
4. SVM using linear kernel takes the maximum time to train and predict the data.
5. While K-Nearest Neighbors is quite simple to execute and effective in predicting accurate results, the predicting process is quite slow.
6. The experiments with SqueezeNet and Vgg16 illustrate that model complexity needs to be at par with the data complexity to avoid both underfitting and overfitting.

5 Conclusion

In this paper, we conduct research on how different algorithms perform in classifying hand-drawn doodles. Experiments with 9 different algorithms were carried out and the Logistic Regression classifier was considered to be the baseline. Most of the algorithms, with the exception of Random Forest and Transfer Learning, performed better than the Logistic Regression classifier in terms of accuracy. Using a Convolutional Neural Network with Adam optimizer gave the best result. The algorithms were evaluated using Confusion Matrices and Accuracy. Future Scope includes extending the research to accommodate Reinforcement learning using Markov's decision process as well as Actor critic. Although past papers suggest that the accuracy does not change much while using Reinforcement Learning and Neural networks still stand to perform better, it would provide us with an insight into the working of the models. 15 classes were used for this project; increasing the number of classes will give a clearer picture in determining the performance of the algorithms in terms of scalability. Evaluation Metric such as Mean Average Precision or plotting CMC curves could also prove to be useful for comparing the models.

References

1. K. Guo, J. WoMa, and E. Xu, "Quick, Draw! Doodle Recognition," p. 6.
2. The Quick, Draw! Dataset. Google Creative Lab, 2020.
3. N. M. Phu, C. Xiao, and J. Muindi, "Drawing: A New Way To Search (Computer Vision)," 2018.
4. M. Lagunas and E. Garces, "Transfer Learning for Illustration Classification," Spanish Computer Graphics Conference (CEIG), p. 9 pages, 2017, doi: 10.2312/ceig.20171213.
5. J. Kim, B.-S. Kim, and S. Savarese, "Comparing image classification methods: K-nearest-neighbor and support-vector-machines," in Proceedings of the 6th WSEAS international conference on Computer Engineering and Applications, and Proceedings of the 2012 American conference on Applied Mathematics, Harvard, Cambridge, Jan. 2012, pp. 133–138, Accessed: Apr. 05, 2020. [Online].
6. D. Ha and D. Eck, "A Neural Representation of Sketch Drawings," arXiv:1704.03477 [cs, stat], May 2017, Accessed: Apr. 06, 2020. [Online]. Available: http://arxiv.org/abs/1704.03477.
7. E. D. Monica, P. Davu, C. P. Caroline, and D. J. Jagannath, "Doodle Recognition using machine learning for hearing and speech-impaired people," in 2019 2nd International Conference on Signal Processing and Communication (ICSPC), Mar. 2019, pp. 109–112, doi: 10.1109/ICSPC46172.2019.8976846.
8. A. Bosch, A. Zisserman, and X. Munoz, "Image Classification using Random Forests and Ferns," in 2007 IEEE 11th International Conference on Computer Vision, Oct. 2007, pp. 1–8, doi: 10.1109/ICCV.2007.4409066.
9. X. Zhang, X. Li, Y. Liu, and F. Feng, "A survey on freehand sketch recognition and retrieval," Image and Vision Computing, vol. 89, pp. 67–87, Sep. 2019, doi: 10.1016/j.imavis.2019.06.010.