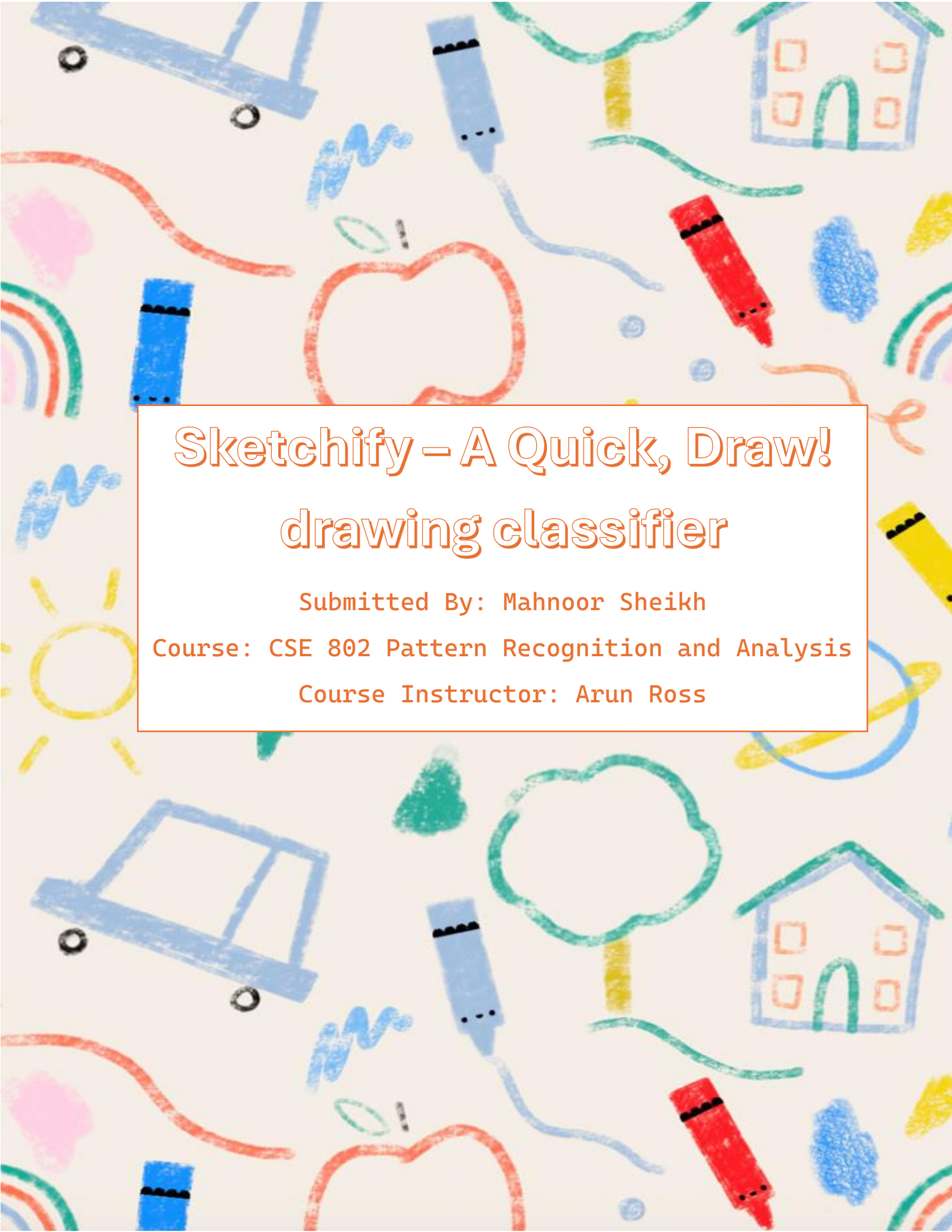


Sketchify – A Quick, Draw! drawing classifier

Submitted By: Mahnoor Sheikh

Course: CSE 802 Pattern Recognition and Analysis

Course Instructor: Arun Ross

TABLE OF CONTENTS

Problem Statement

Teaching machines to recognize freehand sketches is both a scientific challenge and a playful application of modern AI. Online drawing games exemplify this. When played against humans in a multiplayer setting, the outcome is dependent on the creativity and intuition of the human mind. However, when played against a machine, the performance of the underlying machine learning model is put to the test. One such game is Google Creative Lab's *Quick, Draw!*, which is developed to train a neural network to recognize doodling. This serves as the inspiration for this project.

The objective is to develop machine learning models that can classify hand-drawn digital drawings or doodles by identifying the object or category they represent. For this purpose, three Bayesian classifiers, a collection of classical models (Logistic Regression, Support Vector Machine, K-NN, XGBoost), and a Recurrent Neural Network (RNN) have been trained on varying sample sizes. The following sections outline the preprocessing, feature engineering and model evaluation steps, providing insights into each model's performance.

Data Overview

The *Quick, Draw!* dataset contains 50 million drawings contributed by over 15 million players across 345 categories. For this project, 1000 random drawings from 10 categories each have been sampled, namely 'apple', 'baseball', 'bridge', 'circle', 'cow', 'flower', 'moustache', 'speedboat', 'square', and 'yoga'.

Categories 'apple, baseball, circle, flower, square' display low intraclass and moderate interclass variations; whereas 'bridge, cow, moustache, speedboat, yoga' exhibit high intraclass and interclass variations.

Since the raw dataset is so vast, a preprocessed version has been used. The simplified drawing files constitute a simplified drawing vector with:

- drawing aligned to the top-left corner to have minimum values of 0
- uniform scaling into a 256x256 region (to have a maximum value of 255)
- resampled strokes with 1 pixel spacing
- simplified strokes using the Ramer–Douglas–Peucker algorithm with an epsilon value of 2.0
- timing information removed

It includes the following features: *word* (category of the drawing), *countrycode* (player's country), *timestamp* (time the drawing was created), *recognized* (whether the drawing was classified by the model), *key_id* (unique identifier across drawings), and the *drawing array* (JSON array representing the vector drawing with x and y pixel coordinates).

For illustration purposes, the drawing array is of the form:

```
[
[ // First stroke
[x0, x1, x2, x3, ...],
[y0, y1, y2, y3, ...]
],
```

```
[ // Second stroke
[x0, x1, x2, x3, ...],
[y0, y1, y2, y3, ...]
],
... // Additional strokes
]
```

Initial Data Analysis (IDA)

I. Data Cleaning

Since the dataset only contains individual stroke coordinates for each drawing, features need to be manually extracted. To prepare for this, the drawing array is further simplified.

- Each ndjson line (row) is parsed to extract the class label and the drawing array.
- The stroke data is converted into an array of shape $(total\ points, 3)$, where columns 1 and 2 store the x and y pixel coordinates, and column 3 acts as a marker for the end of a stroke (0 by default and 1 at the final point of each stroke).
- x and y coordinates are scaled [0,1] range, so all doodles share the same spatial range. This avoids introducing bias for drawings with coordinates spanning a larger range and ensures each drawing is assigned an equal weight.
- Deltas are computed for the pixel coordinates, so column 1 now stores the difference between consecutive points for x coordinates and column 2 stores the difference between consecutive points for y coordinates.
- Redundant first row is dropped since no deltas exist for the first point and the final array has shape $(total\ points - 1, 3)$.

These steps are implemented on each sample and the resulting cleaned dataset has the shape (10000, 2) with 10 classes.

For illustration purposes, the preprocessed drawing array is of the form:

```
[
[dx0, dy0, m0],
[dx1, dy1, m1],
... // Additional strokes
]
```

II. Feature Engineering

59 features are extracted from the preprocessed drawing array. These encompass:

Stroke Movement features: *mean_dx, mean_dy, std_dx, std_dy, max_dx, max_dy, min_dx, min_dy, num_strokes, total_points, avg_stroke_len, trajectory_len, longest_stroke, shortest_stroke, ratio_longest_shortest, var_stroke_lengths, avg_jump_distance, std_jump_distance*

*‘dx’ and ‘dy’ are the deltas computed earlier.

*‘jump_distance’ is the spatial distance between stroke-end and next stroke-start.

Statistical features: *skew_dx, skew_dy, kurtosis_dx, kurtosis_dy, q25_dx, q75_dx, q25_dy, q75_dy*

Geometric/Spatial features: *bbox_width, bbox_height, bbox_area, bbox_perimeter, bbox_diagonal, aspect_ratio, centroid_x, centroid_y, start_to_centroid, end_to_centroid, avg_distance_to_centroid, std_distance_to_centroid*
*'bbox' refers to the bounding box for an image.

Convex Hull features: *hull_area, hull_perimeter, solidity*
*convex hull is the smallest polygon that encloses all the stroke points.
*'solidity' refers to the ratio of the drawing's filled pixel area to the total hull_area.

Angular/Curvature features: *total_angle_change, mean_segment_angle, std_segment_angle, max_angle_change, min_angle_change, avg_curvature, max_curvature, std_curvature*
*'segment_angle' measures the angle between two consecutive points.

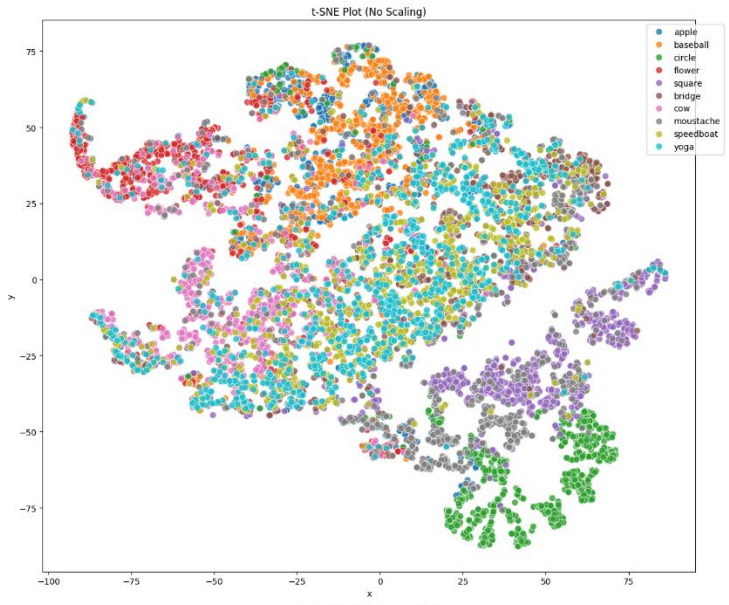
Other features: *dominant_frequency, hu_1, hu_2, hu_3, hu_4, hu_5, hu_6, hu_7, fractal_dimension, straightness*
*'dominant_frequency' captures the dominant back-and-forth pattern/motion repeated in the overall drawing.
*Hue Moments summarize the overall form of the drawing that don't change if you move, resize, or rotate it, making them invariant features. Each Hu moment represents different weighted sums of pixel positions that, when combined, represent the drawing's global outline.
*'fractal_dimension' measures how detailed/complex/twisty the doodle is by counting the number of boxes of different sizes needed to cover all the strokes.
*'straightness' represents if strokes were straighter or wandering.

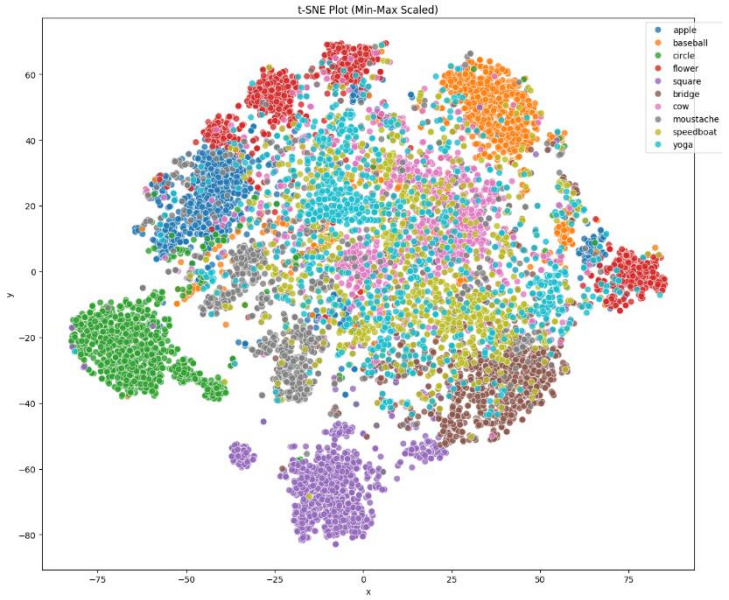
Finally, the dataset results in shape (10000, 60) with 10 classes. All features have numerical data types and have no missing values.

III. Data Normalization

To convert all numerical features to the same range to avoid model bias, Z-score normalization and Min-Max scaling are employed. Z-score normalization centers each feature around 0 with a unit variance and Min-Max scaling transforms all feature values to the [0,1] scale. Silhouette scores and t-SNE plots are used to conclude the most suitable scaling method for our data, hence we compare the results for unscaled, Z-normalized and MinMax scaled datasets.

Silhouette scores indicate how well-clustered the data points are, with a high score being preferred. Unscaled data gives the highest score of 0.478, followed by MinMax scaled data (0.162), and Z-normalized displays the lowest score of 0.084. This indicates that unscaled data results in distinguishing clusters, however, this information may not be accurate since the score is calculated using the distance metric and the larger numerical ranges of unscaled data can dominate the distance calculation. Hence, t-SNE plots are further used to visualize the underlying structure. These use two-dimensional plots that emphasize local relationships and preserve neighborhood structures.





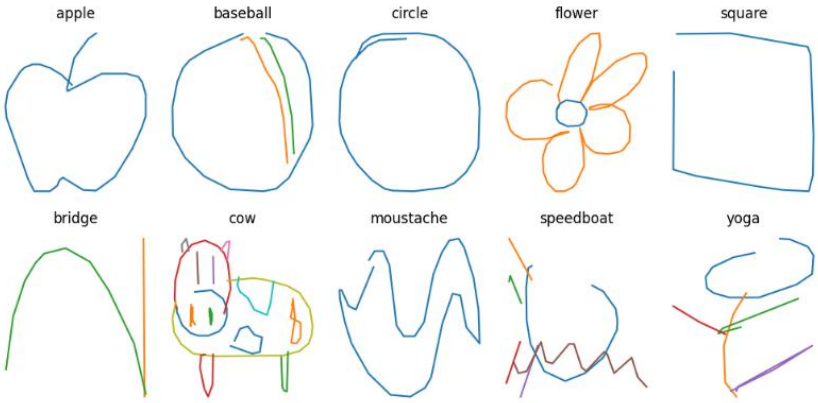
As observed, unscaled data does not result in better class separability. In comparison, the other methods seem more effective. Z-score normalized data shows slightly improved separability than the MinMax scaled data. This may be because each feature is given equal weight in terms of deviation from its mean.

Z-score scaling can also be beneficial since it preserves the relative differences in spread between features which will be useful for PCA and K-NN clustering methods and resolves centering issues for models like logistic regression and SVM.

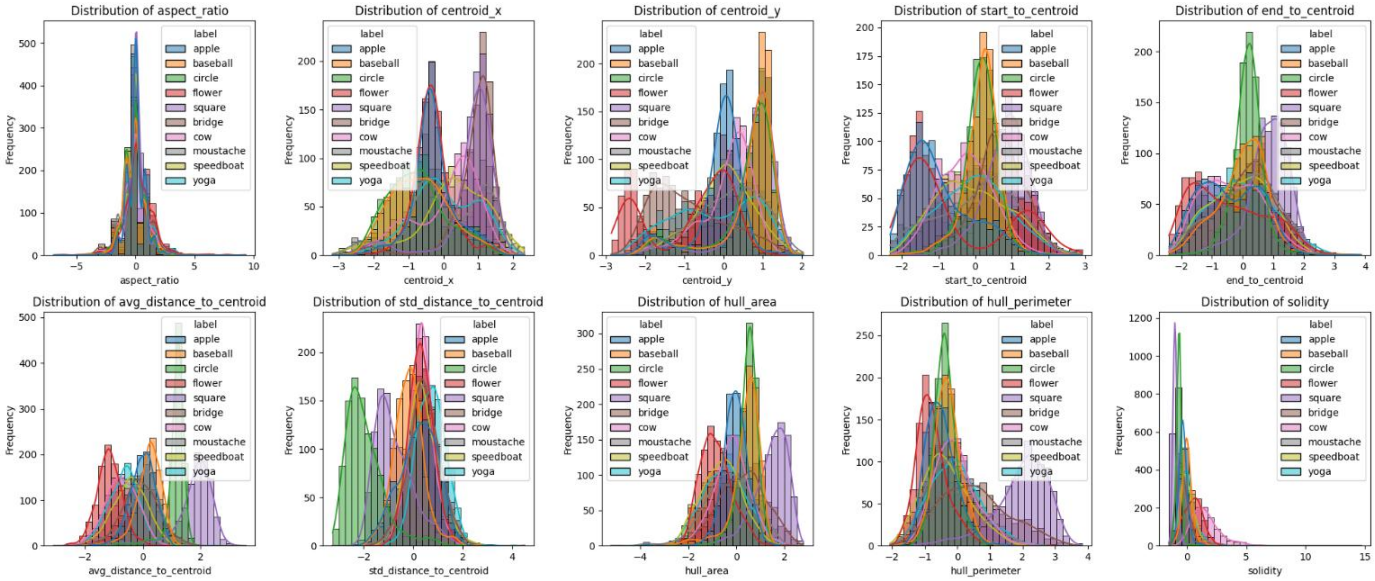
Based on these evaluations, further modeling and analysis has been conducted on Z-score normalized data.

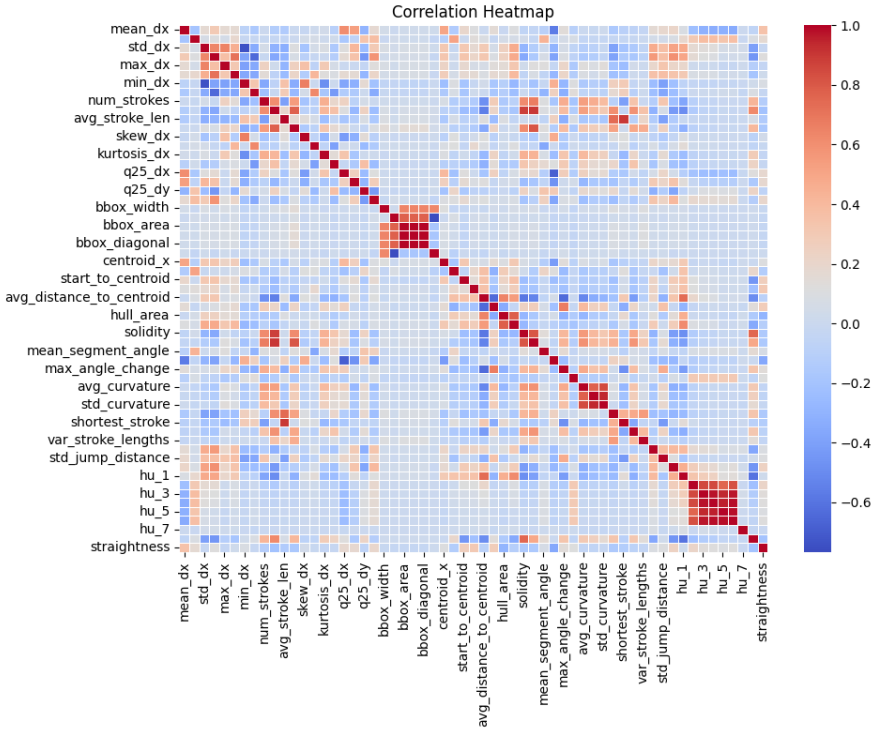
Exploratory Data Analysis (EDA)

These images are randomly sampled doodles from each class. Since they're hand-drawn by people with varying artistic skills (some questionable), they may not perfectly represent their intended class label.



Histograms for each feature across all classes reveal that most features have unimodal distributions, with some representing gaussian distributions and some showing a more skewed structure. All feature distributions depict high overlap among classes. The plot below displays histograms for some features. The full plot for all features can be found in the code file.





High correlation is observed among the hue moments (as expected). Curvature and geometric features based on the bounding box also display high correlation. This can also be expected since these have been derived from similar underlying variables. The remaining features display moderate to low correlation. Since high correlation could lead to multicollinearity issues and interfere with model stability, dimensionality reduction techniques will be applied ahead.

Predictive Modeling

The dataset is split into train, test, and validation sets. This is done for three varying proportions to evaluate classifier performance on differing training sample sizes. The three splits are 70-15-

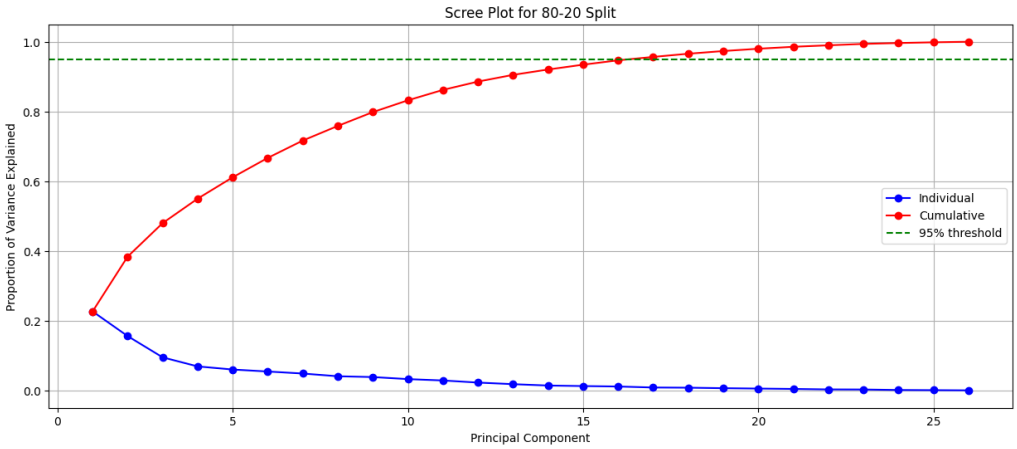
15, 80-10-10, and 90-5-5 (train-test-validation). This translates to training sample sizes of 7000, 8000 and 9000, respectively. All classes are roughly equally represented in all training sets.

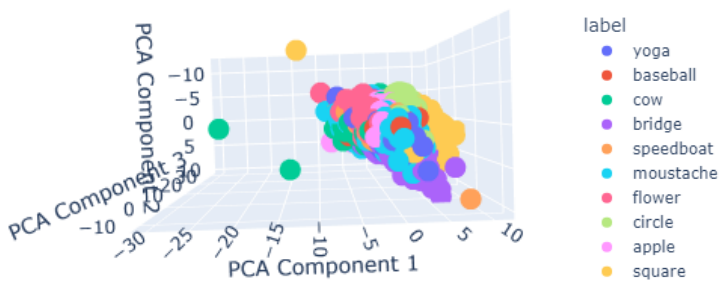
I. Feature Extraction

The ANOVA (Analysis of Variance) test evaluates if the mean of a feature differs across multiple classes. It is revealed that all features are statistically significant in distinguishing at least one class from others. Hence, no features are dropped.

Sequential Forward Floating Selection (SFFS) algorithm with logistic regression classifier is implemented for each training set. An upper bound of the desired number of features is not assigned. Instead, 3-fold cross-validation is used to find the best performing feature subset. The number of features selected for each training set are 23, 26, 25 in order, with minor differences. SFBS was not selected as the feature selection method since it is more effective when a significant number of features are non-informative and +l -r was not employed since we did not have many correlated feature blocks to choose the (l,r) thresholds.

Since the number of features is still high, Principle Component Analysis is applied with the 95% variance rule. PCA creates uncorrelated principal components from correlated features and is applied post the train-test split to prevent data leakage. The scree plot for 80% training set is displayed below. It can be observed that 16 principle components are needed to capture 95% of the variance in the dataset. Similarly, 14 components are needed for 70% training set and 15 for the 90% one.

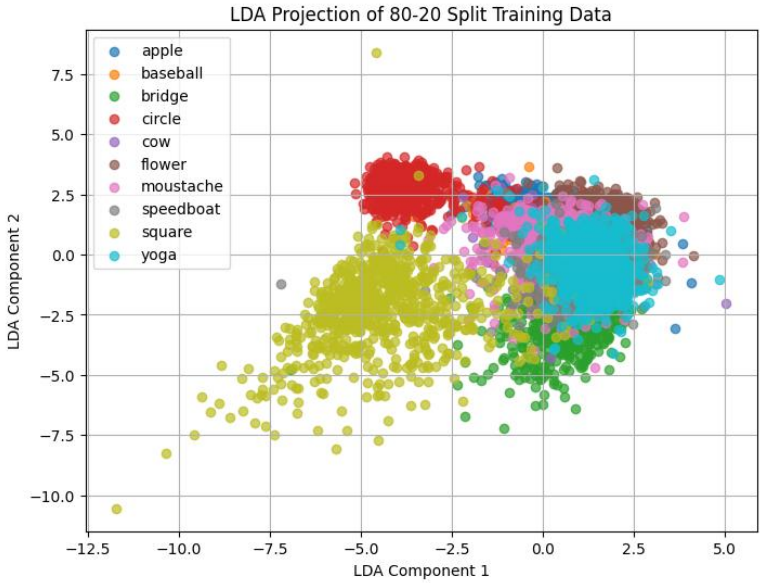




It can be observed from the 3-dimensional plot that class separability is still not achieved when the data is visualized using the first three principle components.

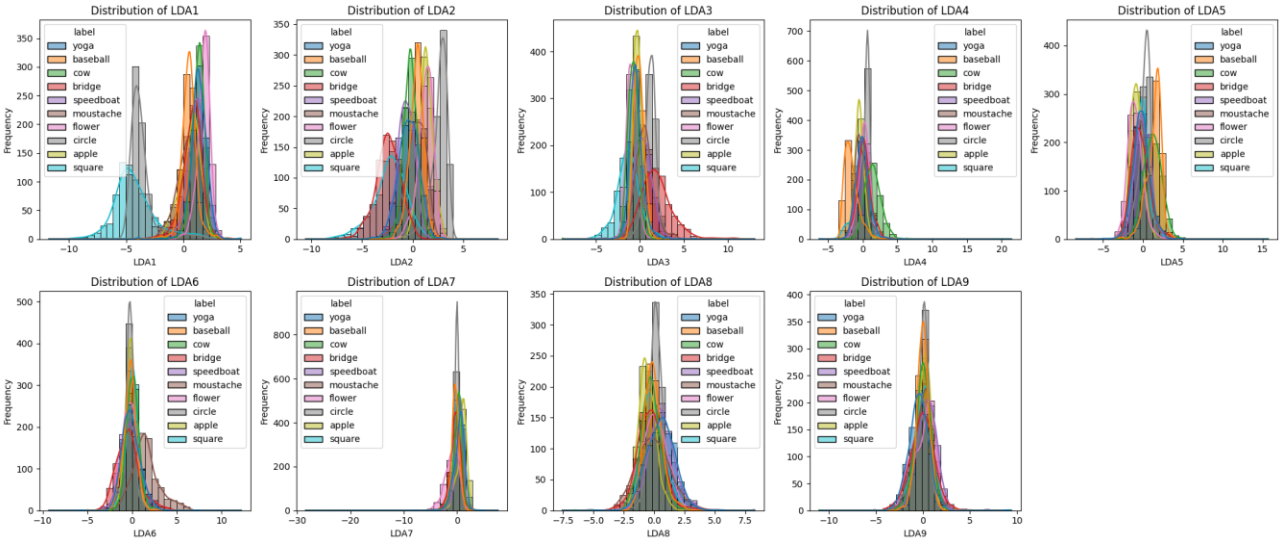
The last step dimensionality reduction technique employed is Multiple Discriminant Analysis (MDA) for feature projection to a further lower dimension.

MDA finds linear discriminants that maximize separation between classes by maximizing between class variance and minimizing within class variance. MDA is applied on each training set and the dataset is reduced to 9 components. The plot below shows MDA class separability using first two MDA components. It can be seen that the classes are now more distinguishable.



By using this dimensionality reduction approach, noise and redundancy is removed via PCA and a feature space that is optimized for class separability is created using MDA. Features also appear roughly bell-shaped across classes, fulfilling assumptions of normally distributed variables needed for some models ahead.

The transformed datasets are now ready for model training.



II. Modeling and Results

Bayesian Classifiers (based on maximum a posteriori principle)

Class-conditional PDFs estimated assuming a Multi-Variate Gaussian density function

This model assumes the likelihood function $p(\underline{x}|\omega_j) \sim N(\underline{\mu}_j, \underline{\Sigma}_j)$ follows a multivariate gaussian distribution $p(\underline{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{[-\frac{1}{2}(\underline{x}-\underline{\mu})^t \Sigma^{-1}(\underline{x}-\underline{\mu})]}$ for every feature. Mean, correlation matrices

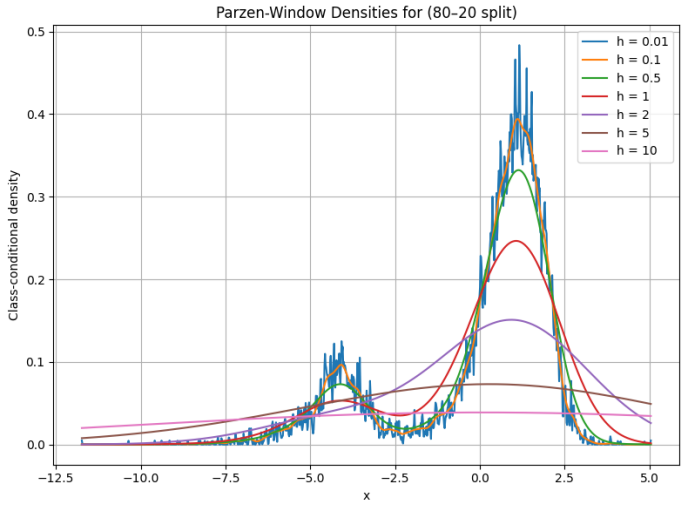
and the prior distribution $P(\omega_j)$ for each class are computed from the training samples. The posterior probability is computed via $P(\omega_j|\underline{x}) = p(\underline{x}|\omega_j)P(\omega_j)$.

Class-conditional PDFs estimated assuming features are independent, and every feature can be modeled using a Gaussian

This model assumes the likelihood function $p(\underline{x}|\omega_j) \sim N(\underline{\mu}_j, \underline{\sigma}_j{}^2)$ follows a univariate gaussian distribution $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{[-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}]}$, with unknown mean and variance parameters. This means the features are uncorrelated and the covariance matrices are diagonal matrices. Since the training samples are assumed to possess the iid property, the parameters are estimated using Maximum Likelihood Estimation (MLE). The prior distribution and posterior probability are computed the same way as before.

Class-conditional density values of test samples estimated using the Parzen-Window non-parametric density estimation scheme with Spherical Gaussian kernel

This model makes no assumptions about the distribution and parameters. The dimensionality reduction pipeline tends to produce uncorrelated, scaled features with weak covariance, resulting in symmetric normal distributions. Hence, a spherical gaussian kernel $\varphi(x) = \frac{1}{(2\pi)^{4.5}} e^{[-\frac{1}{2}(\underline{x})^t(\underline{x})]}$ is best suited for these properties. The density values are estimated using $p_n(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h^9} \varphi[\frac{x-x_i}{h}]$. Seven window width values h = [0.01, 0.1, 0.5, 1, 2, 5, 10] are evaluated on each training set using 5-fold cross-validation, and h = 0.5 is concluded the best for all. This is then used to make the final predictions. The prior distribution and posterior probability are computed the same way as before.



As h increases, local properties are lost, and the global shape is retained. As h decreases, density estimations display jagged behavior, with local properties retained and global shape disturbed. This may be the reason for h=0.5 being an optimal window width, since it provides a balance between the two extremes. However, it can be observed from the plot that local properties like bimodal distribution are lost for the first feature with this choice of the window width.

Model	Accuracy	Precision	Recall	F1-score	Best Pred Class	Worst Pred Class
Train-Val-Test Split: 70-15-15						
1	75.60%	0.75	0.75	0.75	circle, square	speedboat, yoga
2	72.93%	0.73	0.73	0.73	circle, square	speedboat, yoga
3	77.20%	0.77	0.77	0.77	circle, square	speedboat, yoga
Train-Val-Test Split: 80-10-10						
1	76.60%	0.77	0.77	0.77	circle, square	speedboat, yoga
2	75.30%	0.75	0.75	0.75	circle, square	speedboat, yoga

3	77.70%	0.78	0.78	0.78	circle, square	speedboat, yoga
Train-Val-Test Split: 90-5-5						
1	73.80%	0.74	0.74	0.73	circle, square	speedboat, yoga
2	73.00%	0.73	0.73	0.73	circle, square	speedboat, yoga
3	76.60%	0.77	0.77	0.77	circle, square	speedboat, yoga

Non-parametric Bayesian classifier using Parzen-Window consistently achieves the highest accuracy across all train splits. This is followed by the Multivariate Gaussian Bayesian classifier. As the training set fraction increases from 70% to 90%, the overall average empirical accuracy and F1-score tend to increase. The 80% split for the training data displays the best model stability and highest performance, suggesting that too small training sample sizes may overfit the data.
Classes 'circle' and 'square' stand out across all models with recall>0.88 and F1-score>0.90. Due to their simplistic structure, they have the lowest intraclass variation making them easily distinguishable. This is also applicable to classes 'flower', 'baseball', and 'apple' with F1-score 0.84-0.85 approximately. 'speedboat' and 'yoga' are the weakest performers with F1-score≈0.55, due to high intraclass and interclass variations. They are often confused for 'cow', 'moustache' or 'bridge' categories.

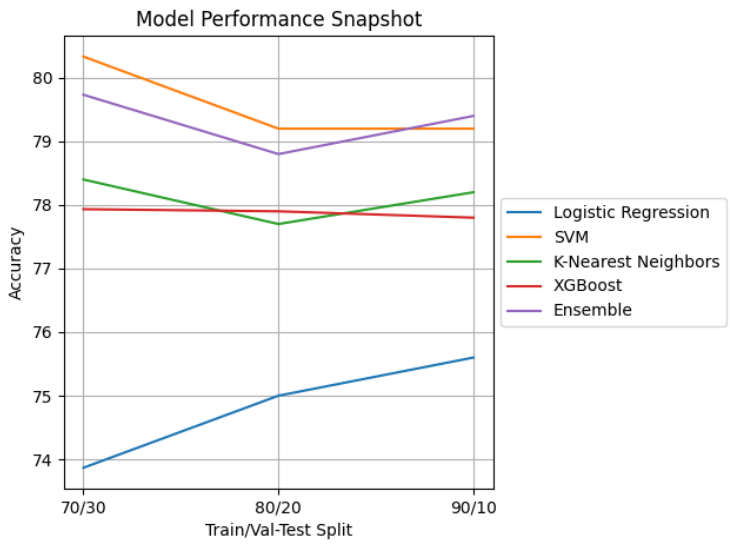
Traditional Classifiers

Four classical models have been trained on each dataset. Logistic regression is selected for a robust linear approach. SVM with RBF kernel is chosen to handle possible non-linear boundaries in the MDA transformed low dimension. K-NN classifier is selected to capture local decision boundaries. XGBoost is chosen to handle complex relationships without heavy parametric assumptions. The ensembled model is created to evaluate how well the models perform when combined.

Each model has been fine-tuned using 5-fold cross-validation on the validation set to identify the best subset of hyperparameters from regularization strength, nearest neighbors, maximum depth, and learning rate. Models are then retrained using the best parameters on the training set and evaluated on the test set. The ensembled model assigns weight to each model based on the validation accuracy computed during cross-validation. The final prediction for a test sample is the class with the maximum weighted vote.

Model	Best Params	CV-Accuracy	Accuracy	Precision	Recall	F1-score	Best Pred Class	Worst Pred Class
Train-Val-Test Split: 70-15-15								
Logistic Regression	'C': 10	0.7153 ± 0.0192	73.87%	0.73	0.74	0.73	circle, square	speedboat, yoga
SVM	'C': 10	0.7840 ± 0.0249	80.33%	0.81	0.80	0.80	circle, flower	speedboat, yoga
K-NN	'n_neighbors': 15	0.7693 ± 0.0232	78.40%	0.79	0.78	0.78	circle, square	speedboat, yoga
XGBoost	'learning_rate': 0.2, 'max_depth': 5	0.7667 ± 0.0241	77.93%	0.78	0.78	0.78	circle, square	speedboat, yoga
Ensemble	Model Weights: Log Regression: 0.2357 SVM: 0.2583		79.73%	0.80	0.80	0.80	circle, square	speedboat, yoga

	K-NN: 0.2535 XGBoost: 0.2526							
Train-Val-Test Split: 80-10-10								
Logistic Regression	'C': 10	0.7320 ± 0.0337	75.00%	0.75	0.75	0.74	circle, square	speedboat, yoga
SVM	'C': 1	0.7880 ± 0.0298	79.20%	0.80	0.79	0.79	circle, square	speedboat, yoga
K-NN	'n_neighbors': 15	0.7810 ± 0.0297	77.70%	0.78	0.78	0.77	circle, square	speedboat, yoga
XGBoost	'learning_rate': 0.1, 'max_depth': 7	0.7760 ± 0.0312	77.90%	0.78	0.78	0.78	circle, square	speedboat, yoga
Ensemble	Model Weights: Log Regression: 0.2379 SVM: 0.2561 K-NN: 0.2538 XGBoost: 0.2522		78.80%	0.79	0.79	0.79	circle, square	speedboat, yoga
Train-Val-Test Split: 90-5-5								
Logistic Regression	'C': 1	0.7420 ± 0.0519	75.60%	0.75	0.76	0.75	square, baseball	speedboat, yoga
SVM	'C': 10	0.7860 ± 0.0403	79.20%	0.80	0.79	0.79	square, baseball	speedboat, yoga
K-NN	'n_neighbors': 20	0.7740 ± 0.0338	78.20%	0.79	0.78	0.78	circle, square	speedboat, yoga
XGBoost	'learning_rate': 0.1, 'max_depth': 7	0.7880 ± 0.0492	77.80%	0.78	0.78	0.78	circle, square	speedboat
Ensemble	Model Weights: Log Regression: 0.2401 SVM: 0.2544 K-NN: 0.2505 XGBoost: 0.2550		79.40%	0.80	0.79	0.79	circle, square	speedboat, yoga



Across all train set splits, accuracy peaks for SVM and the ensembled model. The plot indicates that increased training data can improve learning, but smaller validation sets may undermine model tuning, and hence, the performance.

'circle' and 'square' remain the best performers with precision, recall ≥ 0.91, followed by 'flower' and 'baseball'. 'speedboat' and 'yoga' remain the weakest performers with F1-score≈0.50, frequently misclassified as 'cow', 'bridge', 'baseball', 'moustache', or each other (most frequent).

Recurrent Neural Network (RNN)

This model is developed with inspiration from the RNN tutorial for Quick, Draw! (TensorFlow, n.d.). My approach is a *pytorch* implementation of the tutorial. The model uses the cleaned preprocessed drawing array developed earlier. The features extracted are not used here, instead the sequential stroke data is used to train the neural network to identify recurring patterns. Due to high storage requirements and computational complexity of RNNs, the model is tested on a single 60-20-20 train-val-test split. This split is chosen to have a balance between training and test sets, and have a large enough validation set for effective model tuning.

The model architecture constitutes three 1D convolutional layers that feed into a 2–3-layer bidirectional LSTM, and a single linear output layer to predict the drawing class. The ReLU activation function is employed for all hidden layers. Hyperparameters are fine-tuned using the single hold-out validation set, which are concluded as {'conv_filters': 128, 'lstm_units': 128, 'num_lstm_layers': 2, 'dropout_rate': 0.3, 'learning_rate': 0.001}.

Confusion Matrix:

	apple	baseball	bridge	circle	cow	flower	moustache	speedboat	square	yoga
apple	172	8	0	8	4	0	7	0	1	0
baseball	1	182	0	0	2	0	5	4	0	6
bridge	0	1	179	1	1	1	6	8	1	2
circle	7	4	0	178	2	0	6	1	1	1
cow	1	6	1	0	168	0	13	5	2	4
flower	2	0	0	1	7	179	5	0	0	6
moustache	5	1	3	2	9	0	164	9	3	4
speedboat	1	6	4	0	9	1	14	152	5	8
square	0	1	3	5	2	0	4	3	178	4
yoga	3	2	6	3	23	12	11	14	2	124

RNN achieves the highest accuracy (83.8%) in comparison to all the models. It also outputs the best overall precision, recall, and F1-score of 84%. Classes ‘square’, ‘bridge’, ‘flower’, ‘circle’, ‘baseball’, and ‘apple’ achieve F-1 score ≥ 0.88, with precision, recall≈ 0.90 making them the best distinguishable categories. ‘yoga’ and ‘speedboat’ remain the worse performers, with the former misclassified as ‘speedboat’, ‘baseball’, ‘flower’, ‘cow’, and the latter usually confused with ‘yoga’, ‘moustache’, ‘bridge’, and ‘cow’. ‘cow’ itself also sees significant spillover into ‘yoga’, ‘moustache’ and ‘speedboat’.

Conclusion

The most distinguishable categories are simple, closed-form objects with low intraclass and moderate interclass variations, i.e. ‘apple, baseball, circle, flower, square’. They all achieved F1-score>0.88, with ‘circle’ and ‘square’ displaying almost perfect classification. ‘speedboat’ and ‘yoga’ suffer the most confusion (most frequently misclassified as each other, and ‘cow’, ‘bridge’, ‘flower’, or ‘moustache’) due to their high intraclass and interclass variability.

Bayesian classifiers (non-parametric using Parzen-Window being the best one) served as good baseline models with lower accuracies but identical class distinguishability results as the more advanced models. SVM and the ensemble model achieve comparable results on the dataset. However, when the partitioning exercise is repeated multiple times (across the three splits),

ensemble (20.69% ± 0.1491), K-NN (21.90% ± 0.0867), XGBoost (22.12% ± 0.0032) classifiers show the lowest variance in error rate, versus higher fluctuations for logistic regression (25.18% ± 0.5165) and SVM (20.42% ± 0.2854), underscoring the ensemble's superior stability of balancing lower error rate and variance, making it the more robust choice. RNN classifier is the best performing model, however, since it autonomously learns complex patterns in the dataset that may not have been captured during manual feature extraction and selection.

Although the dataset is roughly balanced, highly variable classes (like 'speedboat', 'yoga', 'moustache', 'cow') behave like imbalanced ones and degrade the accuracy. This suggests that more targeted feature extraction is needed for complex doodles/drawings.

Acknowledgement

Code: https://github.com/mahnoorsheikh16/Sketchify-A-Quick-Draw-drawing-classifier

Google Creative Lab. (n.d.). *Quick, Draw!* [Web site]. Quick, Draw!. Retrieved April 30, 2025, from https://quickdraw.withgoogle.com/ (Quick, Draw!)

Google Creative Lab. (2025, March 11). *The Quick, Draw! Dataset* [GitHub repository]. GitHub. Retrieved April 30, 2025, from https://github.com/googlecreativelab/quickdraw-dataset?tab=readme-ov-file#projects-using-the-dataset

TensorFlow. (n.d.). *Recurrent Neural Networks for Drawing Classification* [Markdown file]. GitHub. Retrieved April 30, 2025, from https://github.com/tensorflow/docs/blob/master/site/en/r1/tutorials/sequences/recurrent_quickdraw.md

Google Cloud. (n.d.). *quickdraw_dataset* [Data set]. Google Cloud Storage. Retrieved April 30, 2025, from https://console.cloud.google.com/storage/browser/quickdraw_dataset;tab=objects?pageState=(%22StorageObjectListTable%22:(%22f%22:%22%255B%255D%22))&prefix=&forceOnObjectsSortingFiltering=false

Google Creative Lab. (n.d.). *Quick, Draw! The Data* [Data set]. Quick, Draw!. Retrieved April 30, 2025, from https://quickdraw.withgoogle.com/data (Quick, Draw! The Data)