ECS271 Machine Learning and Discovery Project Report - Quick-draw Doodler

Brandon Pardi, Prasannadatta Kawadkar, Robin Martin
Department of Computer Science
University of California, Davis, CA 95618
Email: {bmpardi, ppkawadkar, obmartin}@ucdavis.edu

***Abstract*—Here we present `QuickDraw-Doodler`, a variational autoencoder (VAE) designed to incorporate and explicitly model the temporal information as well as spatial in hand-drawn sketches. Unlike much of the previous work, which has been focused mainly on constant progressing vector representations, our method introduces an additional temporal dimension (Δt) to capture the dynamic speed of movement in humans' drawing patterns. Such a temporal aspect allows the model to learn not only what is drawn but how it's drawn. By integrating a Mixture Density Network (MDN) that accounts for both spatial and temporal dynamics, our model captures finer-grained motion characteristics and can generate more temporally coherent sequences. We employ teacher forcing for stable training, as well as a variational framework that leverages latent space sampling for diverse and robust sketch generation. Furthermore, a lightweight classifier is trained for sketch category recognition for efficient and accurate classification. We outline a framework for sketch generation and recognition, and delineate the methodology of our data processing, model and training.**

Introduction

Much of the preexisting work in generative modeling has been applied to modeling pixel based images. However, humans hold a deeper representation of the world than a grid of rgb values. We learn to associate more abstract shapes and features to represent the world around us. These abstract features we learn are more than just an end result of pixels on a screen, as we learn to depict, and even communicate what's in our minds via hand-drawn sketches. These sketches can be thought of as sequential vectors, that by themselves may be meaningless, but when combined in the right way, can be a method of communication. The "QuickDraw Doodler" project aims to learn this representation that humans have understood from a young age. Our model is a novel extension to the existing SketchRNN [1] from Google. It is a similar architecture, however we have adapted it to model temporal aspects of these handwritten sketches on top of spatial aspects. Previously only spatial modeling was done on the sequences, losing valuable insight into the way humans draw doodles. Our goal it to gain this temporal insight that has been lost in previous work, while retaining the ability to learn spatial information as well. We also designed an efficient Convolutional Neural Network (CNN) to classify the end result of doodles with high accuracy, where the challenge lies in balancing model performance with computational efficiency.

The novel contribution we make in this paper is a model architecture that is able to capture and learn the spatial *and* temporal pattens of free hand drawn sketch sequences. In the following sections we detail the dataset, methodology, results, and key findings of these contributions, as well as discussing future directions of this project.

Related Work

SketchRNN introduced the idea of using a recurrent neural network coupled with a latent variable model to generate vector sketches. Subsequent works have incorporated attention, hierarchical structures, and disentangled representations. However, explicit modeling of temporal variance—treating the duration or timing of each stroke as a distributional parameter—has remained underexplored. Our work extends the SketchRNN pipeline by incorporating a time-distribution parameter into the MDN decoder, allowing the model to capture how drawing speed and stroke timing vary across different objects and artistic styles.

Dataset

For training the CNN, we used Google's simplified dataset to classify images. However, for our RNN, we needed to process the raw data from scratch in order to incorporate the temporal component.

The original dataset format was Nx5, where each vector represented (x, y, p1, p2, p3), with x and y being the coordinates, and p1, p2, and p3 representing pen states. For our RNN, the data format was modified to Nx6, adding a time component (t) so that each vector now represents (x, y, t, p1, p2, p3). This change allowed us to capture the temporal dependency of the doodles, as the model needed to learn not just the spatial arrangement but also the progression of strokes over time.

As a result, every step of the data processing pipeline had to be rewritten to account for this new structure. This included adjustments to how the data was normalized, how deltas were calculated, and how the sequences were padded to align with the new time dimension.

- **Full**: Designed for the generative model, this structure retains temporal information (Δt) and stores sketches as sequences of $(\Delta x, \Delta y, \Delta t, p)$. All values are normalized and converted into relative positions and times, enabling the model to learn both temporal and spatial dynamics. The Full dataset contains three classes, each with 50,000 samples.

- **Simplified**: Temporal data is removed, and the vector data is rasterized into 2D arrays representing the original 255x255 pixel sketches. This structure includes 10 classes, with 10,000 samples per class.
- **Reduced**: Derived from the Simplified dataset, images are downscaled to 28x28 pixels using antialiasing, preserving key features while minimizing computational complexity. The Reduced dataset also includes 10 classes, with 10,000 samples per class.

By incorporating temporal information and reorganizing the dataset for diverse tasks, our modifications allow for a more detailed analysis of how drawings evolve over time, addressing a limitation of the original Quick, Draw! dataset. These changes facilitate extensive evaluations of both sketch generation and classification.

GENERATIVE MODEL ARCHITECTURE

Our model is a generative framework designed to produce vector-based sketches by synthesizing pen strokes in sequence. Unlike conventional approaches that focus mainly on the spatial aspects of strokes, our model explicitly incorporates a temporal dimension. Each stroke is represented not only by its spatial displacement (dx, dy) and pen states, but also by a temporal increment (dt) that captures the duration between strokes. By integrating this temporal component into both the latent representation and the output distribution, our model can generate more dynamic and temporally coherent sketches. In what follows, we detail the model's architecture, including the encoder, the latent variable, and the mixture density network (MDN)-based decoder that predicts distributions over (dx, dy, dt) and pen states at each time step.

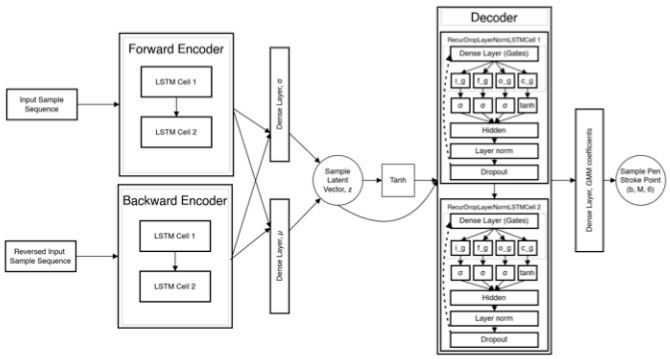

Fig. 1. Schematic representation of the sequence-to-sequence variational autoencoder (VAE) architecture for modeling stroke-based data. The encoder consists of a bidirectional LSTM, with forward and backward pathways processing the input and reversed input sequences, respectively. The hidden states are passed through dense layers to compute the mean (μ) and standard deviation (σ) of the latent space distribution. A latent vector (z) is sampled using the reparameterization trick and transformed with a $\tanh$ activation. The decoder reconstructs the sequence using a hierarchical recurrent architecture with custom recurrent dropout layer normalized LSTM cells. Each cell includes a dense layer for gate values that are then split into separate gates and activated. The next hidden state receives layer normalization and dropout for regularization. The final output layer generates the parameters of a Gaussian Mixture Model (GMM), which predicts pen stroke points, incorporating both spatial coordinates, temporal values, and pen state probabilities.

A. Encoder

The encoder is a bidirectional LSTM that processes the sequence $\mathbf{X}$ and produces a latent representation. We concatenate the last forward hidden state and the first backward hidden state:

$$\mathbf{h}_{\text{enc}} = [\overrightarrow{\mathbf{h}}_T; \overleftarrow{\mathbf{h}}_1] \in \mathbb{R}^{2H}.$$

From $\mathbf{h}_{\text{enc}}$, we derive the parameters of a latent Gaussian distribution:

$$\boldsymbol{\mu} = W_\mu \mathbf{h}_{\text{enc}} + \mathbf{b}_\mu, \quad \log \boldsymbol{\sigma}^2 = W_\sigma \mathbf{h}_{\text{enc}} + \mathbf{b}_\sigma.$$

A latent vector $\mathbf{z} \in \mathbb{R}^Z$ is sampled using the reparameterization trick:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

B. Latent Vector and The Decoder

After obtaining the latent vector $\mathbf{z}$ from the encoder, we condition the decoder on this global context. Initially, we map $\mathbf{z}$ into hidden and cell states for the decoder LSTM. Unlike a simple linear mapping, we now incorporate $\mathbf{z}$ more explicitly by concatenating it with the transformed hidden states:

$$\mathbf{h}_0 = \tanh(W_h \mathbf{z} + \mathbf{b}_h), \quad \mathbf{c}_0 = \tanh(W_c \mathbf{z} + \mathbf{b}_c).$$

We then form the initial LSTM states by concatenating $\mathbf{z}$ with these tanh-activated states:

$$\mathbf{h}_{\text{dec},0} = [\mathbf{h}_0; \mathbf{z}], \quad \mathbf{c}_{\text{dec},0} = [\mathbf{c}_0; \mathbf{z}].$$

This ensures that the latent vector $\mathbf{z}$ is embedded directly into the initial conditions of the decoder, providing a stronger and more persistent global conditioning signal.

At each subsequent timestep t, we form the decoder input by also concatenating $\mathbf{z}$ with the previous output stroke:

$$\mathbf{y}_t = [\mathbf{x}_t; \mathbf{z}],$$

where $\mathbf{x}_{t-1} = (dx_{t-1}, dy_{t-1}, dt_{t-1}, p_{t,t-1}, p_{t,t-1}, p_{t,t-1})$ is the previously generated or ground truth stroke vector (including time and pen states). The decoder then updates its states for the next time step after applying dropout recurrently and layer normalization.

C. Gaussian Mixture Model

The Decoder's output is passed through a mixture density network (MDN) layer to produce parameters for a Gaussian mixture model (GMM) over (dx, dy, dt) along with a categorical distribution over the pen states:

$$\{\pi_{k,t}, \mu^{dx}_{k,t}, \mu^{dy}_{k,t}, \mu^{dt}_{k,t}, \sigma^{dx}_{k,t}, \sigma^{dy}_{k,t}, \sigma^{dt}_{k,t}, \rho_{k,t}\}_{k=1}^{M},$$

$$p_t = (p_{\text{down},t}, p_{\text{up},t}, p_{\text{end},t}),$$

where $\pi_{k,t}$ are mixture weights, μ and σ are means and standard deviations of the Gaussian components, $\rho_{k,t}$ is the correlation between dx and dy, and p_t defines the pen state distribution.

D. Incorporating the Temporal Dimension

As discussed, each stroke in our model is represented by $(dx, dy, dt, p_1, p_2, p_3)$ to capture both the spatial and temporal dynamics of drawing. We opted to keep the spatial distribution modeled together in a bivariate Gaussian distribution with parameters dictated by preceding layers, and model the temporal distribution as a standalone univariate distribution. This decision was motivated due to the potential for spatial deltas to be sampled from an unbounded distribution since negative values are acceptable, while temporal values must remain positive. This allowed us to enforce a constraint unique to dt, in which we parameterize dt through a monotonic transformation of an unconstrained variable. For example, if the decoder predicts a parameter $\hat{dt}$ in $(-\infty, \infty)$, we transform it using the exponential function to ensure $dt = \exp(\hat{dt}) > 0$ at all times. However as we will discuss in the results, this method was not full proof with negative dt values still resulting in the output. We will investigate if this is due to ineffectivity of the method, or a flaw in the codebase.

Classification Model Architecture

We enabled doodle categorization by implementing a convolutional neural network (CNN) architecture, which we call **classifierCNN**, tailored to extract and harness the spatial features inherent in hand-drawn graphics. The proposed architecture has five convolutional layers with an increasing number of filters—32, 64, 128, 256, and 512—where each layer applies 3×3 kernels, a stride of 1, and padding of 1, allowing features to be extracted effectively while preserving spatial resolution. Batch normalization is applied after each convolutional layer to stabilize training and improve convergence, while ReLU activations introduce non-linearity to capture complex patterns.

The model uses max-pooling layers following certain convolutional blocks to decrease the spatial dimensions, retaining only the most salient features while controlling computational complexity. An additional convolutional layer of 256 filters was appended to increase the representational capacity of the model, allowing it to capture fine-grained spatial details. Dropout (set to 30%) is incorporated into the fully connected layers to ensure robustness and prevent overfitting.

The architecture transitions from convolutional layers to two fully connected layers with 512 and 256 neurons, respectively, followed by a final classification layer. The derived spatial features are transformed into high-dimensional embeddings mapped onto predefined doodle categories.

We started with a simple two-layer CNN to evaluate its performance and gradually increased the depth of the network by adding more convolutional layers to improve feature extraction and representational capacity. After experimenting with three, four, and five-layered architectures, we finalized the current five-layer design as it achieved a strong balance between complexity and performance.

To optimize the model, we experimented with varying hyperparameters such as dropout rates, learning rates, and optimizers, and tested different techniques, including max-pooling versus average pooling, and varying filter sizes (3×3, 5×5). Data augmentation techniques such as random rotations, flips, and scaling were also applied to enhance generalization. These efforts helped refine the model into a lightweight and efficient architecture.

While the accuracy of our classifierCNN is slightly lower than Google's deeper 9-layered CNN, our model remains highly competitive and achieves comparable performance. Its streamlined and resource-efficient design makes it particularly well-suited for practical applications where computational resources are limited, providing a robust alternative to deeper architectures.

Loss Functions

The training objective for our model follows the Variational Autoencoder (VAE) framework, which optimizes the Evidence Lower BOund (ELBO) on the marginal likelihood of the data. Conceptually, we want our model to reconstruct the observed sequences of strokes and pen states—including the temporal component—from a learned latent representation while also encouraging the latent space to be well-structured. To achieve this, we combine a reconstruction loss, derived from the mixture density network outputs, with a Kullback–Leibler (KL) divergence term that regularizes the latent distribution.

E. Reconstruction Loss

Our reconstruction loss measures how well the model's predicted distribution over $(dx, dy, dt, p_1, p_2, p_3)$ matches the observed data. At each time step, the decoder outputs parameters of a Gaussian Mixture Model (GMM) for the spatial and temporal increments, and a categorical distribution for the pen states.

For a single timestep t, let

$$\pi_{k,t}, \mu_{k,t}^{dx}, \mu_{k,t}^{dy}, \mu_{k,t}^{dt}, \sigma_{k,t}^{dx}, \sigma_{k,t}^{dy}, \sigma_{k,t}^{dt}, \rho_{k,t}$$

denote the GMM parameters for M mixture components, and let

$$p_{\text{down},t}, p_{\text{up},t}, p_{\text{end},t}$$

denote the pen state probabilities. Given the ground truth stroke (dx_t, dy_t, dt_t) and pen state vector $(p_{t,1}, p_{t,2}, p_{t,3})$, we define the reconstruction loss as the negative log-likelihood:

$$\mathcal{L}_{\text{recon}} = -\sum_{t=1}^{T} \left[\log \left(\sum_{k=1}^{M} \pi_{k,t} \mathcal{N}((dx_t, dy_t, dt_t); \mu_k, \Sigma_k) \right) + \sum_{j=1}^{3} p_{t,j} \log(p_{\text{state},t,j}) \right],$$

where $\mathcal{N}((dx_t, dy_t, dt_t); \mu_k, \Sigma_k)$ is the probability density of the k-th Gaussian component evaluated at the observed stroke increments, and $p_{\text{state},t,j}$ is the predicted probability for the j-th pen state at time t.

Conceptually, this loss encourages the model to assign high probability to the observed strokes and pen states. The mixture model allows flexibility in capturing multimodal stroke distributions, while the categorical term encourages correct

prediction of pen states (e.g., when to lift the pen or end the sketch).

F. Kullback–Leibler Divergence Loss

The latent vector $\mathbf{z}$ is drawn from a Gaussian distribution parameterized by $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ predicted by the encoder. During training, we encourage $\mathbf{z}$ to remain close to a standard Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$. This is achieved by including a KL divergence term:

$$\mathcal{L}_{\text{KL}} = \frac{1}{2} \sum_{j=1}^{Z} \left(\mu_j^2 + \sigma_j^2 - \log \sigma_j^2 - 1 \right),$$

where Z is the dimension of the latent space. Conceptually, the KL term ensures that the latent space does not drift arbitrarily and that it remains smooth and well-structured. Without this constraint, the model might overfit and produce latents that are not generalizable. We also anneal the KL loss in order to focus on reconstruction in the beginning, since that is most important.

G. Overall Objective

The final loss function is the sum of the reconstruction and KL losses:

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \beta \mathcal{L}_{\text{KL}},$$

where β is a weighting factor that can be used to balance latent regularization against reconstruction fidelity. In standard VAE training, $\beta = 1$, but adjusting this factor can help achieve different trade-offs between producing high-quality reconstructions and maintaining a well-structured latent space.

Results

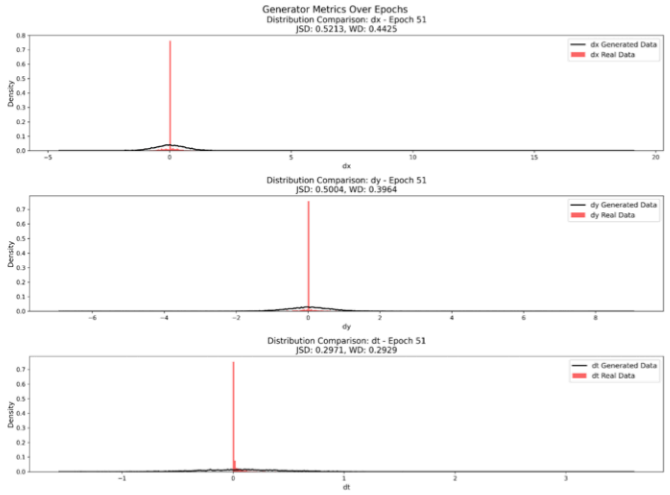

Fig. 2. Distribution comparisons of the generated and real data for dx, dy, and dt at Epoch 51. The dx and dy distributions align closely with the real data, showing good modeling performance. However, the dt distribution exhibits some discrepancies, indicating room for improvement in future work to better capture the temporal dynamics. Metrics such as the Nomralized Jensen-Shannon Divergence (JSD) and Wasserstein Distance (WD) are reported for each variable.

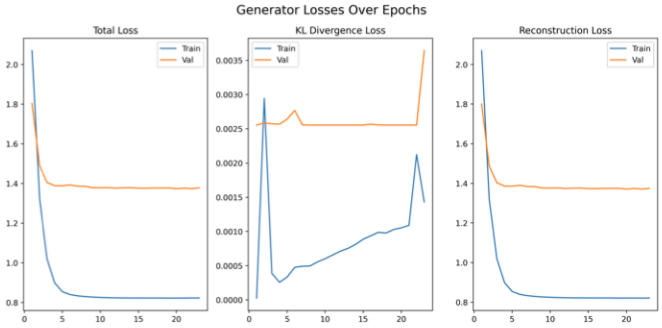

Fig. 3. Generator loss metrics over epochs for training and validation sets. Total loss, KL divergence loss, and reconstruction loss are plotted. The losses generally decrease over time, demonstrating effective learning, though the validation KL divergence exhibits instability. Further refinement of the training process may address this instability.

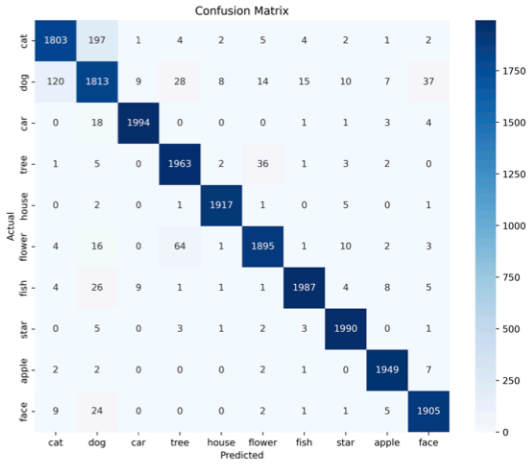

Fig. 4. Confusion Matrix showing approx. 96% accuracy

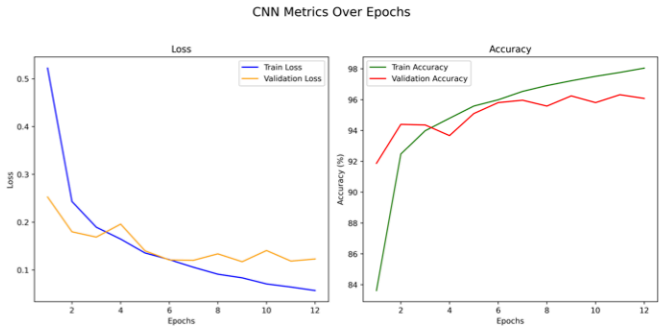

Fig. 5. Train and Validation loss and accuracy for cnn

Applications

Temporal and spatial modeling integration in free-hand sketches shows potential in several fields. In healthcare, the model can decode the drawing of autistic people to provide therapists with insight into emotions and thoughts that may

be difficult to communicate verbally; it also has potential for mental health analysis by recognizing sketch patterns related to emotional indicators such as stress, anxiety, or depression. The system improves accessibility by allowing users with disabilities to communicate and interact using simple drawings as input.

It also has potential benefits for creative industries. Graphic designers can turn initial sketches into professional-level digital products, so efficiency in the creative process will be improved. Game creators can create character and object prototypes, reducing the time spent designing and iterating during the early stages.

Robots with the capability of interpreting human-drawn sketches can communicate and interact better. Additionally, the model can analyze user-generated sketches to uncover trends and patterns in visual representation across different cultures and demographics. This insight provides a opportunity to study human sketching behavior in how people perceive and represent objects through drawings.

Technology also has great value in the preservation and research of cultural heritage. By digitizing and classifying traditional art and symbols from various cultures, it aids in safeguarding and studying these artifacts. Museums can integrate sketch recognition into interactive displays, allowing visitors to engage with historical artifacts through drawing, creating a more immersive and educational experience.

These applications highlight the importance of this technology in addressing real-world challenges, fostering innovation, and promoting accessibility in diverse sectors.

Future Work

Future work for this project would primarily consist of expanding the generator training set to more than a single class. This would allow us to explore the latent space relationship between different classes. It would be interesting to see if we can morph the sketch of one class into another by interpolating points between their respective latent vectors. Additionally, we would like to investigate the full extent the latent vectors represent high-level features by experimenting with latent vector operations of various classes. e.g. can we use the latent vectors to generate a human face onto a pig's body? These are both likely plausible given that the model is trained with more attention to KL loss.

If we can perform latent vector operations, it would also be interesting to explore our model's capability to complete unfinished sketches. By encoding the beginning of a sequence for a sketch, we can then let the model complete the rest of it and see what sort of creations it had come up with. There is also the potential to add a temperature parameter to the GMM sampling to control the model's generative creativity and see what it can come up with. The proposed CNN achieved approximately 96% accuracy on the test set.

Results and Conclusion

Training the Variational Autoencoder (VAE) model presented computational challenges due to its large size, limiting the training to 11 epochs. Although the model did not generate fully coherent doodles, it showed early signs of learning, including the generation of random squiggles. In preliminary tests, we observed some alignment between the real and generated data distributions for both spatial (dx, dy) and temporal (dt) components, with initial metrics indicating some level of correlation. However, the computational constraints prevented us from training the model for longer periods to achieve more coherent results.

We also implemented a Convolutional Neural Network (CNN) using Google's preprocessed dataset, achieving an impressive 96% accuracy on the classification task. In contrast, the VAE was trained on raw input data that we processed to incorporate additional temporal information. While this additional temporal aspect provided valuable insights into the dynamics of human sketches, the limitations in training time hindered the model's ability to fully capitalize on this data.

These results underscore the potential of combining spatial and temporal modeling in generative tasks, while also highlighting the need for further computational resources and longer training durations to achieve optimal results with the VAE. Further work could involve expanding the generator's training set, improving latent vector operations, and experimenting with model enhancements to generate more coherent sketches over time.

Author Contributions

This project was comprised of many different components, below are details of specific contributions from each author:

Brandon Pardi:

- Setup all code and file structure.
- Enums and CLI arguments and handling for parameters and function calling.
- `utils/get_data.py` script:
  - Download simplified data, and scale down with antialiasing.
  - Download raw pen stroke data.
- `utils/image_rendering.py`:
  - Rasterize stroke/image data to display doodles.
  - Animate the doodle strokes to visualize the path of doodles with temporal dependence.
- `utils/process_data.py -> SequentialStrokeData` class:
  - Preprocess data → stroke-6 format.
    - Misc small processing → clamping/filtering/transposing, etc.
    - Z-score normalize sequence deltas.
    - One-hot pen states and add SOS/EOS tokens.
    - Random scale and augment sequences.
    - Sequence padding.
- `models/vae.py -> DoodleGenRNN` class:
  - Train/validation loops for VAE:
    - Kullback-Leibler divergence loss (with annealing).

    - Reconstruction loss.
  - Setup all layers of VAE
  - VAE methods: encode, reparameterize, decode, forward, and latent space conditional sampling.
  - Weight initialization for all VAE layers.
  - All VAE loss metrics logging/visualization code.
  - Approximation and plotting of real and generated data distributions (normalized JSD and Wasserstein distance metrics).
- `models/mdn.py -> MDN` class (initialized as attr of `DoodleGenRNN`
  - Handle Gaussian Mixture Model coefficients from decoder
  - Reconstruction loss methods:
    - Split GMM coefficients from VAE output and apply Tikhonov regularization.
    - Sample from bivariate distribution from GMM parameters for `dx` and `dy`.
    - Sample from univariate distribution from GMM parameters for `dt`.
    - CCE loss for pen state one-hot.
    - Combine them all for reconstruction loss.
- `models/lstm.py -> RecurDropLayerNormLSTM for decoder`:
  - Custom LSTM similar to Torch implementation but with recurrent dropout and layer normalization.
  - Compiled with Torch JIT script for efficiency.

Prasannadatta Kawadkar:

- Designed a 5-layer CNN with increasing filters (32, 64, 128, 256, 512) using 3×3 kernels, ReLU activation, and batch normalization.
- Added max-pooling layers for dimensionality reduction and dropout (30%) for regularization.
- Included two fully connected layers (512 and 256 neurons) and a final classification layer.
- Implemented the CNN architecture in `cnn.py` and created the `train_cnn` function in `train.py` for training.
- Developed `classify.py` for classifying doodles into predefined categories.
- Modified CLI commands to support training and classification using the CNN.
- Experimented with filter sizes (3×3, 5×5), pooling methods (max vs. average), and dropout rates (0.2, 0.3, 0.5).
- Tested different layer depths (2, 3, and 5 layers) and dense layer configurations (256, 512, and 1024 neurons).
- Compared optimizers (Adam, SGD, RMSProp) and chose Adam for faster convergence.
- Preprocessed input images to 28×28 for computational efficiency while preserving features.
- Had major contribution in the final report and presentation for the project.
- Achieved a lightweight and efficient CNN with high classification accuracy for doodles.

Robin Martin:

- Applied the RDP (Ramer-Douglas-Peucker) algorithm to reduce points in curves while preserving shape, improving training speed:
  - Tested multiple epsilon values (e.g., $\varepsilon = 0.2$, $\varepsilon = 1$, $\varepsilon = 2$), balancing curve simplification and shape preservation.
  - Reduced an image of 235 points to 83 with $\varepsilon = 1$ and to around 60 with $\varepsilon = 2$, noting jaggedness at higher simplifications.
  - Set $\varepsilon = 1$ for optimal balance.
- Data Processing and Debugging:
  - Altered sequence padding to match EOS tokens for balancing p3 values.
  - Corrected data normalization inconsistencies by reordering steps to align delta calculations and normalization for accurate formatting.
  - Tested every step with print and/or animate to ensure proper vector sizes and outputs.
  - Noted negative time values after decoding.
  - Removed repeat functions.
- Image Generation
  - Added functions to reverse delta and normalization after decoding to `Animate` function.
  - Updated `generation.py` and `main.py` to integrate `vae.py`
  - Updated `Animate` function to accept `tesnor` values by converting to `.numpy`
  - Updated pen state column selection in `Animate` function to animate generated images of Nx6(expected Nx4).

REFERENCES

[1] David Ha and Douglas Eck. A neural representation of sketch drawings, 2017.