IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DoodleNet: Real-Time Sketch Recognition Using Convolutional Neural Networks

S Adithya Ganesh[1], Sai Shrikar M K[2], Saksham Verma[3]

Dept. of CSE (Emerging Technologies), SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[1]

Dept. of CSE (Emerging Technologies), SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[2]

Dept. of CSE (Emerging Technologies), SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[3]

Abstract: Hand-drawn sketches are abstract, highly variable and difficult for computers to interpret. This paper presents DoodleNet, a compact end-to-end system that recognizes ten categories of hand-drawn objects in real time. A convolutional neural network (CNN) with 872,042 parameters is trained from scratch on 40,000 bitmaps of 28 by 28 pixels from the Google Quick, Draw! dataset and is served through a Streamlit web application whose drawing canvas displays the three most probable classes with confidence scores. A canvas preprocessing routine that crops, scales and centers the user's drawing aligns live input with the training data. The network reaches 92.52% accuracy on 6,000 held-out images, and preprocessing plus prediction take about 62 ms on a single virtual CPU. Controlled tests show that predictions are insensitive to stroke width and ink color, but sensitive to rotation, unreliable on incomplete drawings and over-confident on input that belongs to none of the classes. These limits are reported together with directions for improvement.

Keywords: sketch recognition, convolutional neural network, Quick, Draw!, canvas preprocessing, real-time inference, Streamlit

I. INTRODUCTION

Drawing is one of the oldest and most natural ways in which people communicate, and a quick sketch often explains a concept faster than a paragraph of text. A sketch is deliberately minimal: it keeps only the strokes that identify an object and omits everything else. This abstraction makes sketches fast to produce, but it also makes them hard for a computer to understand. A photograph of a house contains color, texture and fine detail, whereas a sketch of the same house may be only a square, a triangle and a rectangle drawn with a shaky hand, and two people will draw it very differently.

CNNs now classify natural images with accuracy comparable to that of human observers, but sketch recognition has received less attention. Interest has grown with the availability of large crowdsourced collections, the most prominent being Google's Quick, Draw! dataset, which contains about 50 million drawings in 345 categories collected through an online game [1]. Beyond model accuracy, a practical recognizer must cope with extreme intra-class variation, with strokes of different thickness drawn by mouse, trackpad or finger, with incomplete drawings, and with a gap between the curated training bitmaps and a freshly drawn canvas whose scale, position and stroke width are different.

This paper presents DoodleNet, a small end-to-end sketch recognition system built on Quick, Draw! for ten visually distinct categories (apple, bus, cat, dog, fish, flower, house, key, pencil and star). A user draws on a canvas in the browser and, within a fraction of a second, the application shows the three most likely categories with confidence scores. The project covers the full machine learning life cycle: data acquisition, preprocessing, model design, training, evaluation, deployment and user interaction. The main contributions are:

- a compact CNN with 0.87 million parameters, trained from scratch on a CPU-friendly 40,000-image subset;
- a documented six-step canvas preprocessing routine that makes live drawings resemble the training bitmaps;
- a Streamlit application with live top-3 prediction, a view of the model input and an opt-in dataset contribution feature; and
- an evaluation that goes beyond test accuracy to stroke width, ink color, rotation, incomplete drawings, non-class input and latency.

 This work is licensed under a Creative Commons Attribution 4.0 International License

The rest of the paper is organized as follows. Section II reviews related work, Section III describes the data and preprocessing, Section IV presents the model, Section V describes the system, Section VI reports results and limitations, and Section VII gives conclusions and future work.

II. RELATED WORK

Early sketch recognition relied on hand-crafted descriptors and classical classifiers. Eitz et al. introduced a 250-category sketch benchmark and reported a little over 50% accuracy for a bag-of-features SVM, against roughly three correct answers in four for human observers [2]. Deep networks changed this picture: Sketch-a-Net applied a CNN designed for sketch images and reported performance exceeding that of human observers [3], and CNN-based classification of hand-drawn sketches has since been studied further [4]. Recurrent models such as Sketch-RNN instead use the order and direction of strokes and can also generate drawings [5], but they need vector stroke data and are more complex to deploy. Transfer learning from large photograph-trained networks such as VGG [6] and ResNet [7] is possible, but these models expect inputs of 224 × 224 pixels and are heavy on a CPU.

Work on Quick, Draw! itself includes a CS229 course project in which a CNN reached about 60% accuracy over all 345 classes [8], a statistical analysis of the dataset that highlights quality variation between drawings [9], and a public recognition challenge that emphasizes generalization to a held-out distribution [10].

Several practical aspects are rarely documented in a form that can be reproduced end to end: (i) the step from a trained model to an interactive application, (ii) the mismatch between training bitmaps and canvas drawings, (iii) robustness to nuisances such as stroke width, rotation or incomplete input, (iv) the cost of large 345-class models for small demonstrations, and (v) a way to collect new labeled examples from users. DoodleNet addresses these gaps within the scope of a course project.

III. DATA AND PREPROCESSING

A. *Dataset*

The data come from the numpy bitmap version of Quick, Draw! [1], in which every drawing is rendered as a 28 × 28 grayscale image stored as 784 unsigned 8-bit values (0 is background and 255 is a stroke pixel at full intensity). Ten categories were chosen to cover different kinds of familiar objects: apple, bus, cat, dog, fish, flower, house, key, pencil and star. Most pairs are easy to separate, whereas cat and dog share a round head with ears and were expected to be the hardest pair.

Instead of downloading entire class files, the training script requests only the first 4,000 drawings of each class with an HTTP range request (an 80-byte header followed by 4,000 × 784 bytes, about 3 MiB per class and 29.9 MiB in total) and caches them locally, falling back to a full download if the range request fails. This gives 40,000 drawings and makes experiments repeatable on a slow connection.

B. *Training Data Preprocessing*

Pixel values are scaled to the range 0–1 as in (1), reshaped to 28 × 28 × 1 and the labels are one-hot encoded. The samples are shuffled with a fixed random seed (42) and split 85%/15% into 34,000 training and 6,000 test images, which gives about 600 test images per class.

$$x' = x / 255 \tag{1}$$

C. *Canvas Preprocessing*

The most delicate part of the system is converting a canvas drawing into a tensor that looks like a Quick, Draw! bitmap. The canvas returns a 460 × 460 RGBA image with a white background and strokes of a user-selected color and width, whereas the training images are centered, normalized, light strokes on black. The routine performs six steps, illustrated in Fig. 1. (1) *Ink mask:* a pixel is ink if its alpha value exceeds 40 and its gray level is below 225, which is independent of the selected color. (2) *Empty-canvas check:* if fewer than 25 pixels are ink, no prediction is made. (3) *Bounding-box crop* removes the dependence on where and how large the user drew. (4) *Stroke thickening:* if the longer side exceeds 100 pixels, the mask is dilated with a 2 × 2 kernel so that strokes survive the reduction. (5) *Aspect-preserving resize* with area interpolation so that the longer side measures 20 pixels, as in (2). (6) *Centering and contrast normalization:* the result is pasted at the center of a black 28 × 28 image, as in (3), stretched so that the brightest pixel equals 255, and divided by 255.

$$s = 20 / \max(h, w),\ \ h' = \text{round}(s \cdot h),\ \ w' = \text{round}(s \cdot w) \tag{2}$$

$$\text{top} = \text{floor}((28 - h') / 2),\ \ \text{left} = \text{floor}((28 - w') / 2) \tag{3}$$

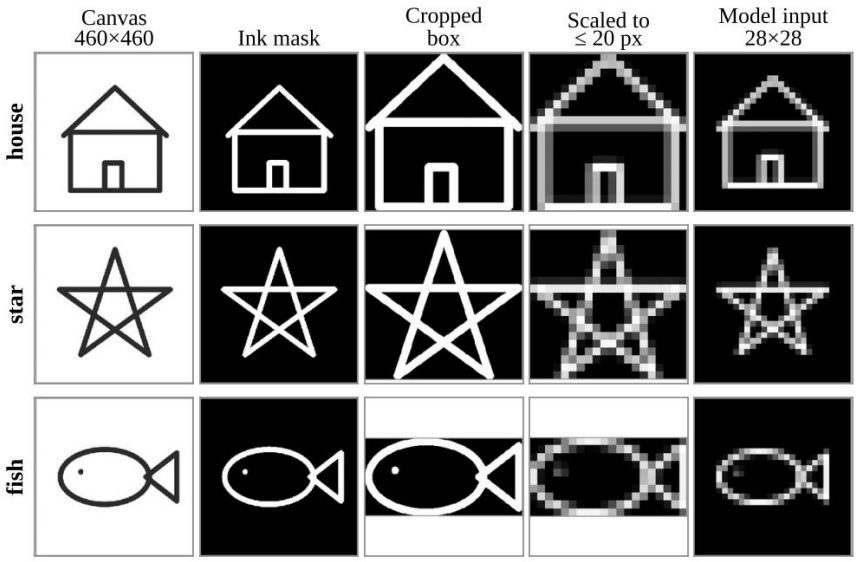


Fig. 1 Stages of canvas preprocessing for a house and a star (generated with the application's own preprocessing routine).

IV. MODEL

A. *Architecture*

The input is a bitmap, so identifying information lies in local patterns such as corners, edges, curves and closed loops. Convolutional layers detect such patterns with few shared weights and pooling makes detection tolerant to small shifts. DoodleNet uses two VGG-style blocks [6], listed in Table I. Pairs of 3 × 3 convolutions with ReLU activation [11] cover the same 5 × 5 area as one 5 × 5 kernel with fewer weights and an extra non-linearity; the filter count doubles from 32 to 64 while the resolution is halved from 28 × 28 to 14 × 14 and then 7 × 7; same padding keeps strokes near the border; batch normalization [12] and dropout [13] (rates 0.20, 0.25 and 0.40) limit over-fitting; and a softmax output allows the top-3 classes and their confidences to be reported directly.

TABLE I LAYER SPECIFICATION OF THE DOODLENET CNN

Layer	Output shape	Parameters
Input	28×28×1	0
Conv 3×3, 32, ReLU + BatchNorm	28×28×32	448
Conv 3×3, 32, ReLU	28×28×32	9,248
MaxPool 2×2 + Dropout 0.20	14×14×32	0
Conv 3×3, 64, ReLU + BatchNorm	14×14×64	18,752
Conv 3×3, 64, ReLU	14×14×64	36,928
MaxPool 2×2 + Dropout 0.25	7×7×64	0
Flatten	3,136	0
Dense 256, ReLU + BatchNorm + Dropout 0.40	256	804,096
Dense 10, softmax	10	2,570
Total		**872,042**

The model has 872,042 parameters (871,338 trainable and 704 non-trainable batch-normalization statistics), about 3.5 MB in 32-bit floating point. The first dense layer holds 92.1% of the parameters, whereas the four convolutional layers hold only 7.5% but account for 18.3 of the 19.1 million multiply-accumulate operations per image. This is small enough for real-time use on an ordinary CPU.

B. *Training*

The network is trained with the Adam optimizer [14] (learning rate 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$) to minimize the categorical cross-entropy in (4), where y is the one-hot label, p the softmax output and N the batch size. Mini-batches of 128 images are used for at most 8 epochs, with a learning-rate reduction (factor 0.5, patience 2) and early stopping (patience 3, best weights restored), both monitoring the validation loss. Training uses TensorFlow and Keras [15], [16]. The optimizer state stored in the saved model confirms 2,128 iterations, which is exactly 8 epochs of 266 steps, and a final learning rate

of 0.0005, so the plateau callback halved the rate once. The hold-out set serves as validation data, so the reported accuracy is slightly optimistic.

$$L = -(1/N)\,\Sigma_n\,\Sigma_k\, y_{n,k} \log p_{n,k} \tag{4}$$

V. SYSTEM DESIGN AND IMPLEMENTATION

A. *Architecture*

The design follows two principles. First, training and inference are separated: the expensive training step runs once offline, and the web application only loads the saved model file and class metadata, as shown in Fig. 2. Second, the preprocessing applied to a live drawing is a single, well-defined function, so that the network sees in deployment what it saw during training. Streamlit [17] re-executes the script on every interaction, so the model is loaded once and cached; preprocessing and prediction, which take a few milliseconds, are repeated on each execution, which lets the display update while the user draws.

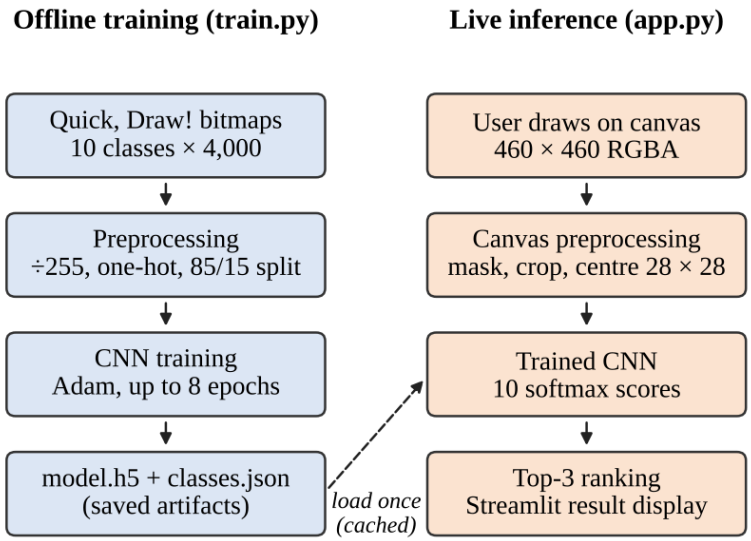


Fig. 2 Offline training pipeline and live inference pipeline of DoodleNet.

B. *Application*

The canvas is provided by streamlit-drawable-canvas [18]: 460 × 460 pixels, free-draw mode, brush width 6–32 pixels and a color picker. Image operations use OpenCV [19] in its headless build. For each drawing the interface shows a highlighted primary-match card and three confidence bars (Fig. 3), together with an expandable panel that displays the 28 × 28 image given to the model. An empty canvas is a normal initial state and a missing model file produces a warning instead of a stack trace. After a drawing is detected, the user may download the 28 × 28 PNG or, after confirming a label, add it to a dataset bank with a manifest file; nothing is stored without this explicit action. The application runs on a CPU without a GPU and is deployed on Streamlit Community Cloud.

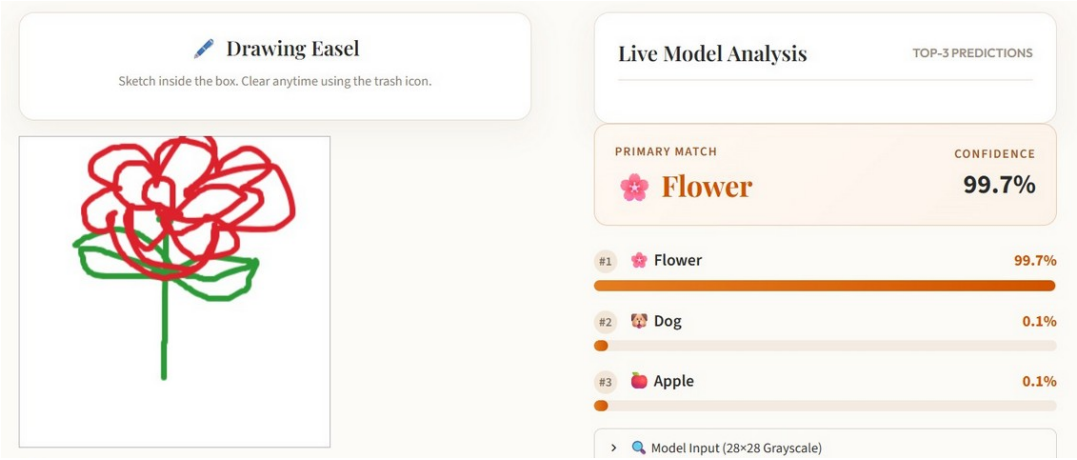


Fig. 3 Main screen of DoodleNet: drawing easel and live top-3 analysis.

 This work is licensed under a Creative Commons Attribution 4.0 International License

VI. RESULTS AND DISCUSSION

The behavioral tests below load the saved model and pass programmatically drawn sketches (460 × 460 canvas, 14-pixel charcoal strokes unless stated) through the same preprocessing function the application uses. They were run in a Linux container with one virtual CPU, TensorFlow 2.21.0 and no GPU.

A. *Test Accuracy*

On the 6,000 held-out images the model reaches 92.52% accuracy, that is 5,551 correct classifications and 449 errors (error rate 7.48%). The normal approximation gives a 95% confidence interval of ±0.67 percentage points (91.9% to 93.2%), against 10% for random guessing. Fig. 4 shows the learning curve. Training accuracy rose from about 79% in the first epoch to about 93% in the last, and training loss fell from about 0.7 to about 0.2. The validation curves were unstable at the start (about 10% accuracy and a loss of about 5.5 in the first epoch, and a dip to about 79% in the fourth), which is typical of batch normalization whose running statistics need a few updates to settle. From the fifth epoch the validation accuracy stayed near 92% and the validation loss near 0.25, within about one percentage point of the training accuracy, so dropout and batch normalization kept over-fitting small.

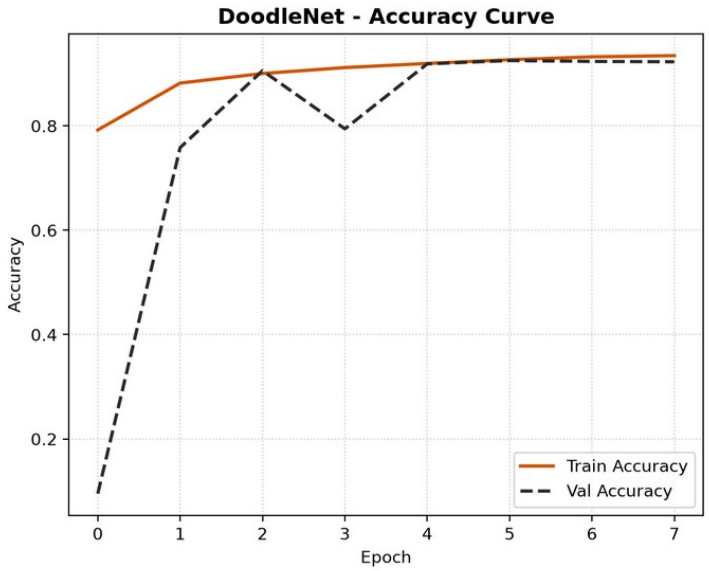


Fig. 4 Training and validation accuracy over the eight training epochs (epochs counted from 0).

B. *Functional Verification*

One clean sketch of each of eight classes was drawn with line, circle and ellipse primitives (cat and dog cannot be drawn faithfully this way). All eight were classified correctly with top-1 confidence between 93.9% and 100% (Table II). The lowest confidence was obtained for the key, which also has the fewest active pixels (76 of 784). These sketches are cleaner than most human doodles, so the result confirms that the pipeline works but is not an estimate of accuracy on real users.

TABLE II TOP-1 PREDICTIONS FOR EIGHT TEST SKETCHES

Drawn	Top-1 (%)	Runner-up (%)	Active pixels
apple	100.0	flower (0.02)	127
bus	100.0	cat (0.00)	169
fish	96.8	apple (1.15)	109
flower	98.4	dog (0.56)	148
house	100.0	pencil (0.00)	184
key	93.9	apple (3.78)	76
pencil	99.7	key (0.29)	127
star	99.5	cat (0.28)	174

C. *Robustness*

Stroke width and ink color. A house drawn with eight stroke widths from 4 to 32 pixels was always classified as house with confidence of at least 99.88%, while the number of active pixels in the model input rose from 122 to 238. A star

drawn in seven colors was classified as star with 99.0% to 99.5% confidence whenever the color's gray level was below the ink threshold of 225 (six of seven colors); only a very pale color (gray level 240) was not detected, and the canvas was then treated as empty.

Rotation. The nearly symmetric star was recognized at every angle from 0° to 180° with confidence of at least 96% (Fig. 5a). The house was recognized at 0° and 10°, but at 20°, 30° and 45° it was classified as dog, fish and pencil, at 90° it was still house but with only 46.4% confidence, and upside down (180°) it was classified as bus with 95.8% confidence. Training drawings are mostly upright, and a CNN is tolerant to small translations through pooling but not to rotation, so augmentation with small random rotations is a worthwhile improvement.

Incomplete drawings. Because the application predicts while the user draws, a house was rendered stroke by stroke (Fig. 5b). The first stroke alone was classified as pencil (91.9%) and the partial drawings with two to four strokes, which contain only a box, as bus (36.0% to 66.4%). Once the first roof line was added as the fifth stroke, the prediction changed to house with 99.96% confidence. Confidence values at the start of a drawing should therefore not be over-interpreted.

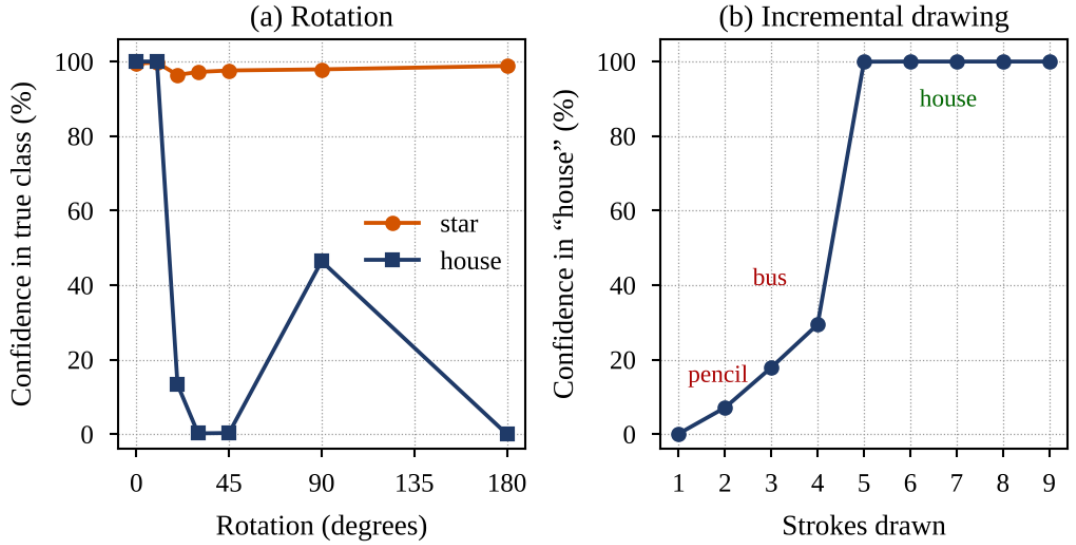


Fig. 5 Confidence in the true class (a) as the drawing is rotated and (b) as a house is drawn stroke by stroke.

D. *Input That Belongs to No Class*

A softmax classifier always distributes all of its probability among the known classes. A random scribble of twelve straight lines was labeled cat with 55.8% and dog with 31.2%, because crossing segments resemble the whiskers and ears of cat and dog doodles, whereas a clean house receives practically all of the probability for a single class. Neural networks are known to be over-confident [20]; practical remedies include rejecting predictions below a confidence threshold [21], calibrating probabilities with temperature scaling [20], or adding an "other" class trained on scribbles.

E. *Inference Latency*

Each pipeline step was timed 300 times after a warm-up (Table III). Preprocessing takes about 1.6 ms, and the model.predict call used by the application takes 60.9 ms on average (95th percentile 73.4 ms), so preprocessing and prediction together take about 62.4 ms. Calling the model directly as a function is about 4.3 times faster because predict carries batching overhead that is unnecessary for a single image. The delay perceived by the user is dominated by the Streamlit script rerun and the transfer of the canvas image between browser and server, not by the model.

TABLE III LATENCY PER SKETCH (300 RUNS, ONE VIRTUAL CPU)

Step	Mean (ms)	Median (ms)	95th pct (ms)
Canvas preprocessing	1.6	1.5	1.8
model(x, training=False)	14.1	13.5	18.8
model.predict(x)	60.9	58.3	73.4

F. *Comparison with the Literature*

A direct numerical comparison is not meaningful because the studies use different datasets, numbers of classes and protocols (Table IV). The result shows the expected relation between difficulty and accuracy: the fewer and more distinct the classes, the higher the accuracy. DoodleNet is about 9.3 times better than random guessing for ten classes, and with

 This work is licensed under a Creative Commons Attribution 4.0 International License

0.87 million parameters it is roughly 30 times smaller than ResNet-50 (25.6 million) [7] and roughly 160 times smaller than VGG-16 (138 million) [6]. It should not be read as a claim to outperform the other studies.

TABLE IV DOODLENET COMPARED WITH REPORTED FIGURES

Study	Task	Reported result
Eitz et al. [2], bag of features + SVM	250 classes, TU-Berlin	A little over 50%
Eitz et al. [2], human observers	250 classes	About 3 in 4 correct
CS229 project [8], CNN	345 classes, Quick, Draw!	About 60%
DoodleNet (this work)	10 classes, Quick, Draw!	**92.52%** (chance 10%)

G. *Limitations*

Only ten of the 345 categories are supported, and the model cannot say that a sketch is "something else." The 28 × 28 bitmap discards stroke order and timing, and the model is sensitive to rotation and unreliable while a drawing is incomplete. The evaluation is limited: the hold-out set was used for both early stopping and the final accuracy, only one training run was made, per-class precision and recall are not reported, the behavioral tests used programmatically drawn sketches, and no study with human users was carried out. Contributed sketches are labeled without verification, may be lost on a host with a non-persistent file system, and are not yet used for retraining.

VII. CONCLUSION AND FUTURE WORK

DoodleNet shows that a compact CNN with 872,042 parameters, trained in minutes on a CPU from 40,000 crowdsourced doodles, can recognize ten kinds of hand-drawn objects in real time, reaching 92.52% test accuracy with about 62 ms of preprocessing and prediction time. Equally important, the experiments show that a single accuracy figure hides weaknesses in rotation, incomplete drawings and unknown input, and that much of a practical system lies outside the model: how a drawing is cropped, scaled and centered, and how failures are handled.

Future work includes (i) training with small random rotations, shifts, scalings and stroke-thickness changes; (ii) a three-way train/validation/test split, repeated runs, per-class metrics and a small set of live drawings from several people; (iii) rejection of unknown input through a confidence threshold, calibration or an "other" class; (iv) more categories; (v) stroke-aware models that combine a CNN with a recurrent network [5]; (vi) persistent storage and label review for contributed sketches so that the model can be retrained; (vii) global average pooling and conversion to TensorFlow Lite or TensorFlow.js for in-browser inference; and (viii) class-activation maps to explain predictions.

ACKNOWLEDGMENT

The authors thank Dr. S. Anitha Jebamani, Assistant Professor (O.G), for her guidance, and Dr. P. Chitra, Professor and Head, Department of CSE (Emerging Technologies), SRM Institute of Science and Technology, Vadapalani Campus, for her encouragement and support.

REFERENCES

[1] J. Jongejan, H. Rowley, T. Kawashima, J. Kim, and N. Fox-Gieg, "The Quick, Draw! A.I. experiment", Google Creative Lab, Mountain View, CA, USA, 2016. [Online]. Available: https://quickdraw.withgoogle.com

[2] M. Eitz, J. Hays, and M. Alexa, "How do humans sketch objects?", ACM Trans. Graph. (Proc. SIGGRAPH), vol. 31, no. 4, pp. 44:1–44:10, 2012.

[3] Q. Yu, Y. Yang, F. Liu, Y.-Z. Song, T. Xiang, and T. M. Hospedales, "Sketch-a-Net: A deep neural network that beats humans", Int. J. Comput. Vis., vol. 122, no. 3, pp. 411–425, 2017.

[4] A. T. Kabakus, "A novel sketch recognition model based on convolutional neural networks", in Proc. Int. Congr. Human-Comput. Interact., Optim. Robot. Appl. (HORA), 2020.

[5] D. Ha and D. Eck, "A neural representation of sketch drawings", in Proc. Int. Conf. Learn. Represent. (ICLR), 2018, arXiv:1704.03477.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", in Proc. Int. Conf. Learn. Represent. (ICLR), 2015, arXiv:1409.1556.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 770–778.

[8] Guo et al., "Quick, Draw! doodle recognition", course project report, CS229, Stanford Univ., Stanford, CA, USA, 2018.

[9] R. Fernandez-Fernandez et al., "Quick, Stat!: A statistical analysis of the Quick, Draw! dataset", 2019, arXiv:1907.06417.

 This work is licensed under a Creative Commons Attribution 4.0 International License

[10] Google, "Quick, Draw! doodle recognition challenge", Kaggle, 2018. [Online]. Available: https://www.kaggle.com/c/quickdraw-doodle-recognition
[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines", in Proc. Int. Conf. Mach. Learn. (ICML), 2010, pp. 807–814.
[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", in Proc. Int. Conf. Mach. Learn. (ICML), 2015, pp. 448–456.
[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", J. Mach. Learn. Res., vol. 15, pp. 1929–1958, 2014.
[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", in Proc. Int. Conf. Learn. Represent. (ICLR), 2015, arXiv:1412.6980.
[15] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems", 2015, arXiv:1603.04467.
[16] F. Chollet et al., "Keras", 2015. [Online]. Available: https://keras.io
[17] Streamlit Inc., "Streamlit documentation" [Online]. Available: https://docs.streamlit.io
[18] F. Andrianasolo, "streamlit-drawable-canvas", GitHub repository. [Online]. Available: https://github.com/andfanilo/streamlit-drawable-canvas
[19] G. Bradski, "The OpenCV library", Dr. Dobb's J. Softw. Tools, 2000.
[20] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks", in Proc. Int. Conf. Mach. Learn. (ICML), 2017, pp. 1321–1330.
[21] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks", in Proc. Int. Conf. Learn. Represent. (ICLR), 2017, arXiv:1610.02136.

 This work is licensed under a Creative Commons Attribution 4.0 International License