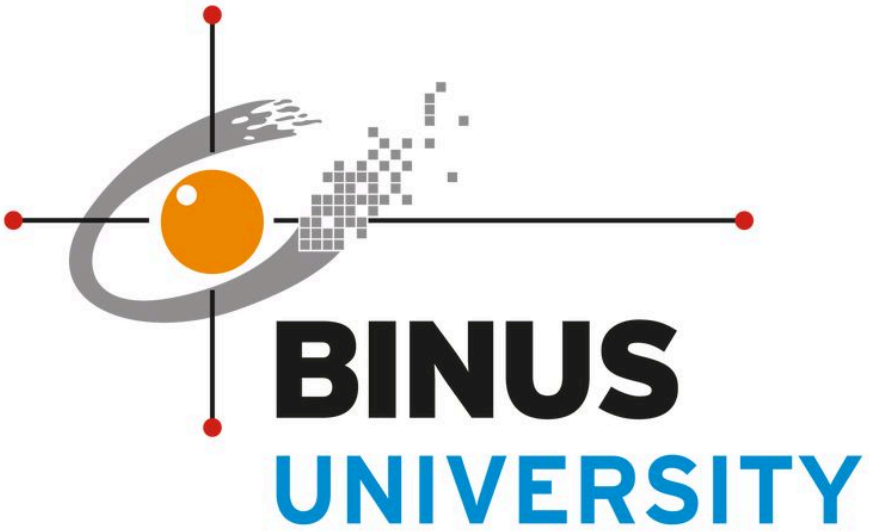
Image Classification of Hand-Drawn Sketches using Convolutional Neural Networks



Deep Learning Final Project - LB01

Group Members:

2702368223 - Helena Aurelia Sanjaya

2702331100 - Reinhart Gian Widjaja

2702210432 - William Sebastian Liman

1. Introduction

1.1. Background

The field of computer vision has made great progress in classifying realistic photos. However, recognizing hand-drawn sketches has not been touched enough compared to photorealistic images. This area has remained challenging because of the vast differences between human drawing styles. Unlike photos, sketches do not have distinct colors and textures. Sketches only rely on simple lines to implicate meaning. Building a model to understand these abstract drawings is the core motivation of this project. Specifically, we aim to develop a deep learning model capable of classifying hand-drawn sketches across the variations of drawing styles and abstraction levels.

1.2. Objectives

The main objectives of this project are:

- To preprocess and prepare a subset of the QuickDraw Dataset for training and evaluation.
- To design and implement a Convolutional Neural Network (CNN) architecture suitable for sketch recognition.
- To train the CNN model and evaluate its performance using appropriate metrics.
- To deploy the trained model for real-time inference of hand-drawn sketches.

2. Dataset

2.1 Dataset Description

The dataset utilized for this project is a subset of the **QuickDraw Dataset**, a massive collection of 50 million drawings across 345 categories, contributed by players of the Quick, Draw! game. For this project, a curated subset of the Google Quick, Draw! dataset was used, consisting of 26 selected object classes to provide a balanced and manageable level of visual diversity. Each class contains 5,000 unique images, resulting in a total dataset of 130,000 images. All images are provided by Google in a preprocessed raster format with a fixed resolution of 255 × 255 pixels, ensuring consistent input dimensions across the dataset. The drawings are represented as sparse grayscale images composed of simple line strokes created by users, making them well suited for sketch recognition tasks. Data collection was performed using a custom fetching script that streams drawings directly from the QuickDraw dataset, filters out duplicates using content-based hashing, and stores an equal number of samples per class to maintain class balance.

2.2 Class Categories

The dataset used in this study consists of 26 distinct object categories selected from the Google Quick, Draw! dataset. These categories represent a diverse set of everyday objects, animals, and symbols, including angel, apple, bat, book, candle, castle, cat, cup, dog, door, fish, flamingo, hexagon, lantern, light bulb, mailbox, necklace, octopus, parachute, pencil, pig, skull, swan, sword, table, and van. Each class contains an equal number of samples to maintain class balance and prevent bias during training. By selecting a fixed and moderate number of categories, the dataset achieves a balance between classification complexity and computational feasibility, enabling the model to learn discriminative visual patterns while remaining tractable for training and evaluation.

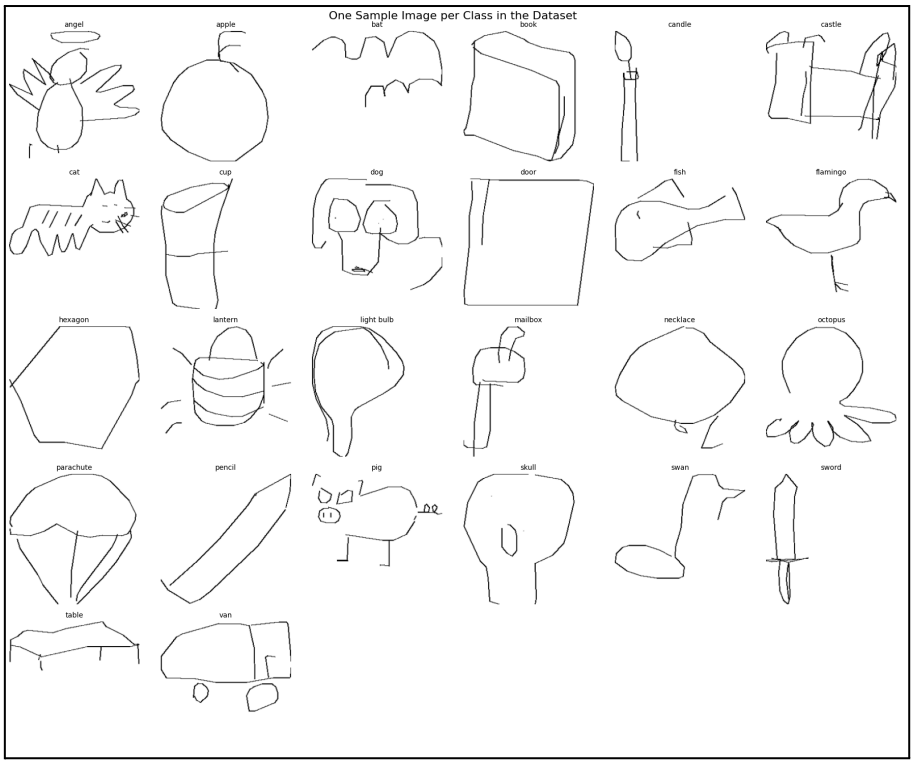


Figure 2.2.1 Sample Image for every Class

2.3 Data Preprocessing and Augmentation

The data preprocessing pipeline begins by loading the rasterized sketch images from the curated Quick, Draw! dataset using a directory-based data loader, where each subdirectory represents a class label. All images are resized to a uniform resolution of 224 × 224 pixels to match the input shape required by the convolutional neural network. The images are then converted from RGB to grayscale, reducing the input channels to one while preserving the essential stroke information. Pixel values are normalized by scaling them to the range [0, 1] to improve numerical stability during training. The dataset is subsequently split into training, validation, and test subsets using a fixed ratio. During training, additional data augmentation techniques, including random horizontal flipping, brightness adjustment, and contrast variation, are applied to improve generalization, while validation and test data undergo only deterministic preprocessing to ensure unbiased performance evaluation.

Figure 2.3.1 Data Augmentation

3. Modeling

A Convolutional Neural Network (CNN) is chosen as the primary modeling approach due to its proven efficacy in various image recognition tasks.

3.1 Model Architecture

The proposed model will be a standard sequential CNN designed for image classification, featuring an architecture that balances complexity and performance. The architecture details are summarized in the table below:

Layer Type	Parameters	Output Shape
Input Layer	-	(224, 224, 1)
Conv Block 1	2× Conv2D (32 filters, 3×3), BatchNorm, ReLU	(224, 224, 32)
Max Pooling 1	Pool Size: (2, 2)	(112, 112, 32)
Dropout	Rate: 0.5 × dropout	(112, 112, 32)
Conv Block 2	2× Conv2D (64 filters, 3×3), BatchNorm, ReLU	(112, 112, 64)
Max Pooling 2	Pool size: (2, 2)	(56, 56, 64)
Dropout	Rate: 0.5 × dropout	(56, 56, 64)
Conv Block 3	2× Conv2D (128 filters, 3×3), BatchNorm, ReLU	(56, 56, 128)
Max Pooling 3	Pool size: (2, 2)	(28, 28, 128)
Dropout	Rate: 0.5 × dropout	(28, 28, 128)
Conv Block 4	2× Conv2D (256 filters, 3×3), BatchNorm, ReLU	(28, 28, 256)
Max Pooling 4	Pool size: (2, 2)	(14, 14, 256)
Dropout	Rate: 0.5 × dropout	(14, 14, 256)

Global Avg Pooling	-	(256)
Dense Layer 1	Units: 256, Activation: ReLU	(256)
Dropout	Rate: dropout	(256)
Dense Layer 2	Units: 128, Activation: ReLU	(128)
Dropout	Rate: dropout	(128)
Output Layer	Units: 26, Activation: Softmax	(26)

Figure 3.1.1. Model Architecture

3.2 Regularization and Optimization Techniques

To enhance generalization and training stability, the model incorporated multiple regularization techniques, including L2 weight decay, batch normalization, progressive dropout, and label smoothing. Batch normalization was applied throughout the network to stabilize gradient flow and accelerate convergence, while dropout was used strategically to reduce overfitting without degrading spatial feature learning. Optimization was performed using the Adam optimizer with adaptive learning rate scheduling, combining exponential decay with ReduceLROnPlateau to enable both rapid initial learning and fine-grained convergence in later stages. Early stopping based on validation loss was employed to prevent overfitting, resulting in efficient training and robust performance on unseen test data.

4. Evaluation

The trained model was evaluated on a held-out test set to measure its generalization across 26 classes. Training was performed from scratch for up to 50 epochs using early stopping and learning rate scheduling, resulting in effective convergence without unnecessary training. The full training process took **160 minutes and 26 seconds** on a single GPU. Final evaluation achieved a test accuracy of **90.13%** with a loss of **1.0267**, indicating strong and stable classification performance.

4.1 Performance Metrics

The primary evaluation metrics will include:

- **Overall Accuracy:** The fraction of correctly classified sketches.
- **Confusion Matrix:** To visualize the per-class performance and identify common misclassifications.
- **Classification Report:** Providing precision, recall, and F1-score for each of the 26 classes.

4.2. Model Performance

Overall, the trained convolutional neural network demonstrates strong and stable performance across training, validation, and test evaluations. The training and validation accuracy curves show consistent improvement with no significant divergence, indicating effective learning without severe overfitting, while the corresponding loss curves steadily decrease and stabilize over epochs. Per-class accuracy on the test set remains high and relatively uniform across the 26 object categories, with an average accuracy exceeding 0.90, suggesting that the model generalizes well to unseen sketches. The normalized confusion matrix further confirms this behavior, as correct predictions dominate the diagonal with limited off-diagonal confusion. Analysis of the top misclassified class pairs reveals that most errors occur between visually similar categories, such as animals or simple objects with overlapping stroke patterns, which is expected in freehand sketch recognition. These results indicate that the model is robust, well-calibrated, and suitable for real-time deployment within the interactive sketch-based web application.

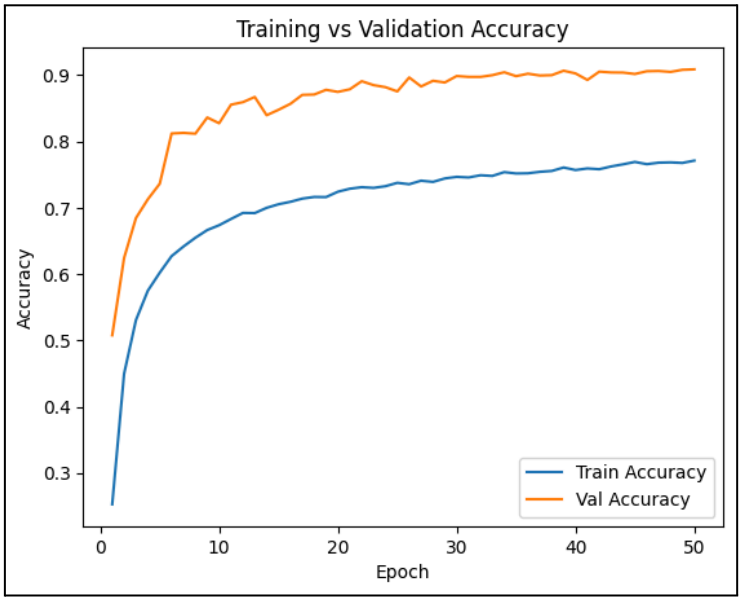


Figure 4.2.1 Train and Validation Accuracy over Epochs

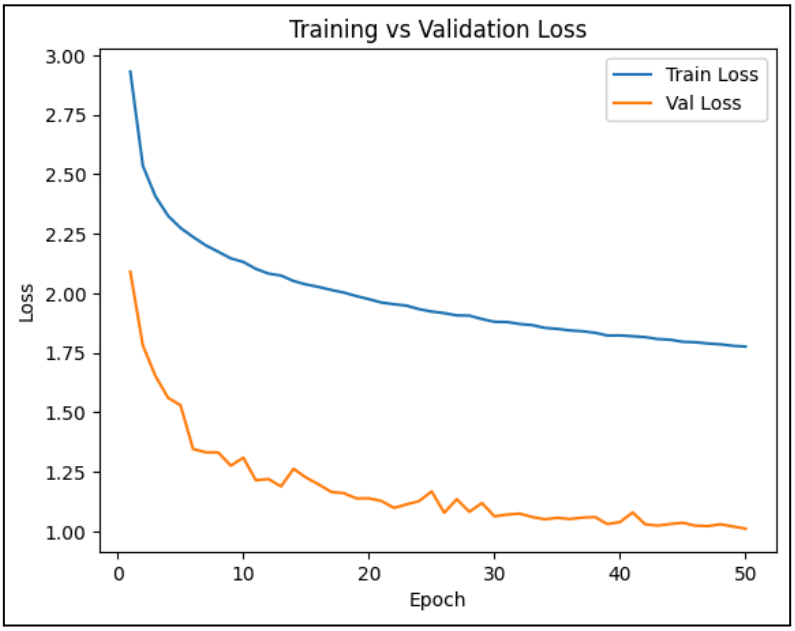


Figure 4.2.2 Train and Validation Loss over Epochs

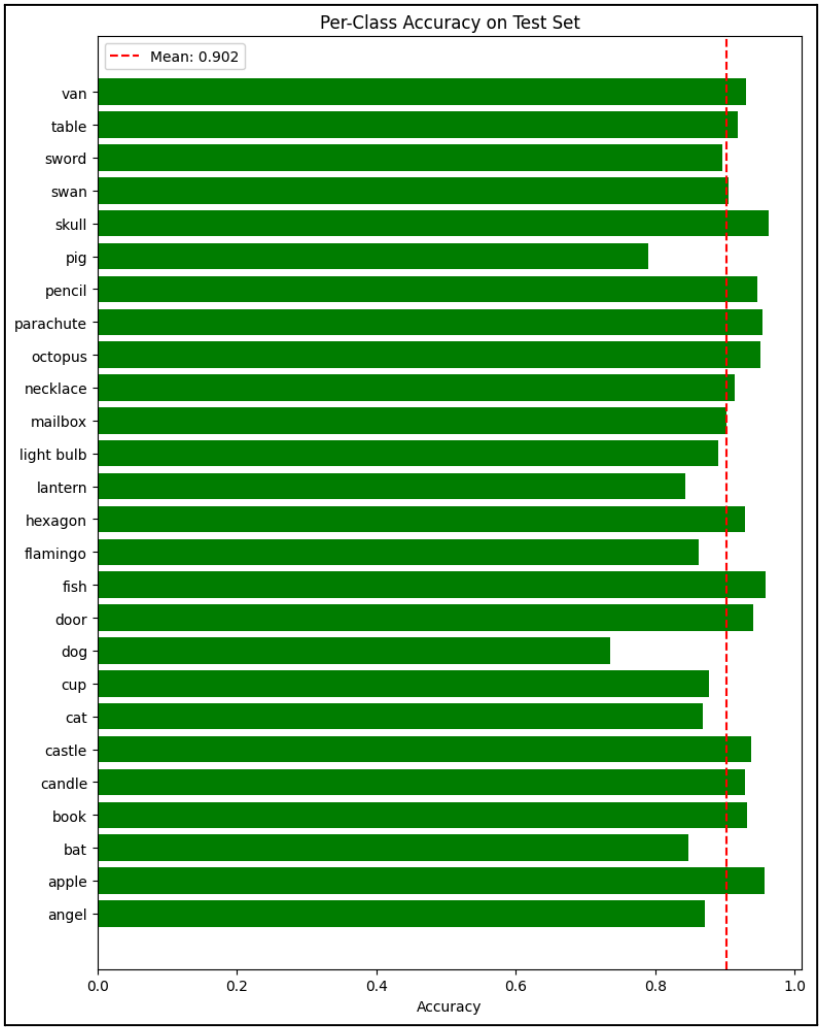


Figure 4.2.3 Per-Class Accuracy on Test Set

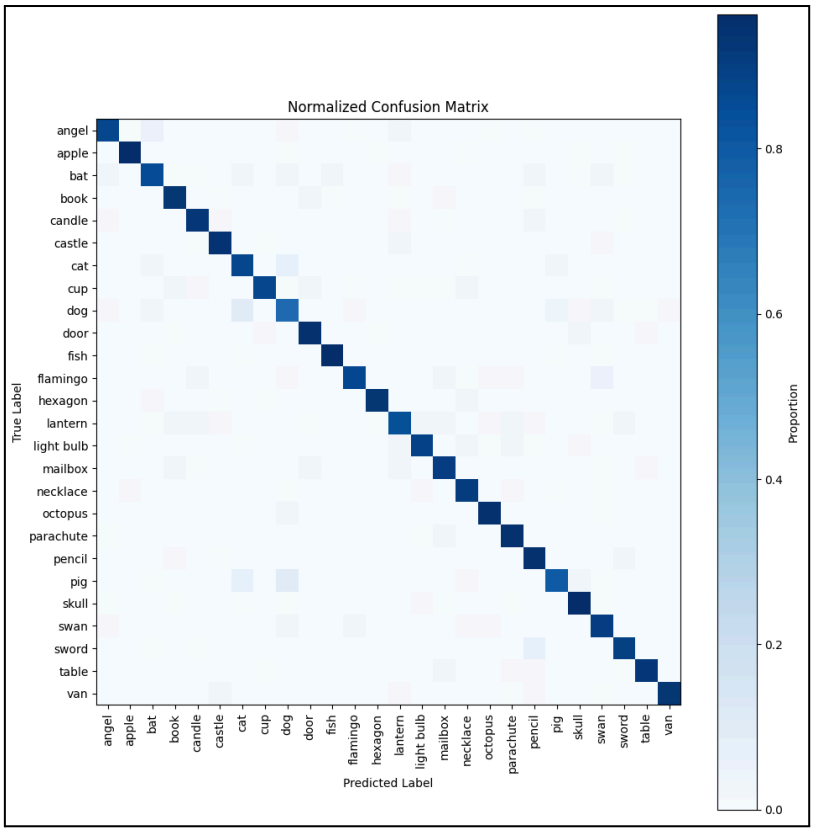


Figure 4.2.4 Normalized Confusion Matrix

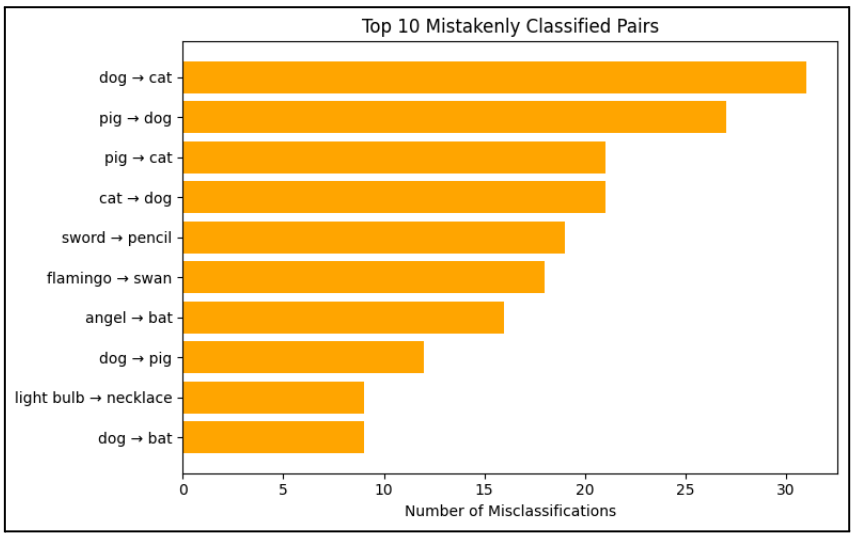


Figure 4.2.5 Top 10 Mistakenly Classified Pairs

5. Deployment

The deployed system includes a browser-based frontend that provides an interactive sketching interface implemented using HTML5 Canvas and React. Users draw sketches directly on the canvas, which dynamically resizes to a maximum resolution of approximately 800 × 600 pixels depending on the screen size. The canvas captures the drawing as a raster image, preserving stroke thickness and spatial structure. To support real-time interaction while minimizing unnecessary requests, image data is transmitted to the backend only after a short inactivity period of 0.5 seconds, ensuring efficient communication without disrupting the user experience.

Once the frontend sends the canvas image to the backend, a preprocessing pipeline is applied to transform the raw user drawing into a format compatible with the trained convolutional neural network. The received image is converted to grayscale and normalized to a pixel value range of [0, 1]. Stroke regions are automatically detected and cropped to remove excess background, after which the image is padded to a square aspect ratio and resized to 224 × 224 pixels to match the model's expected input shape. A channel dimension is then added, resulting in a final tensor of shape (1, 224, 224, 1) suitable for inference. This preprocessing pipeline mirrors the transformations applied during training, ensuring consistency between training and deployment and enabling reliable real-time classification of user-drawn sketches.

Component	Technology	Description
Frontend	React	Provides an interactive drawing canvas where users sketch objects. The canvas captures drawings at approximately 800 × 600 pixels and sends the image to the backend after a 0.5-second inactivity debounce.
Backend (API)	Python (FastAPI)	Receives sketch images from the frontend, applies preprocessing to transform the input into a model-compatible format, and performs real-time inference using the trained CNN model.
Model	Keras/TensorFlow	A trained convolutional neural network used to classify user-drawn sketches into one of 26 predefined object categories.

Figure 5.1.1 Deployment Techstacks

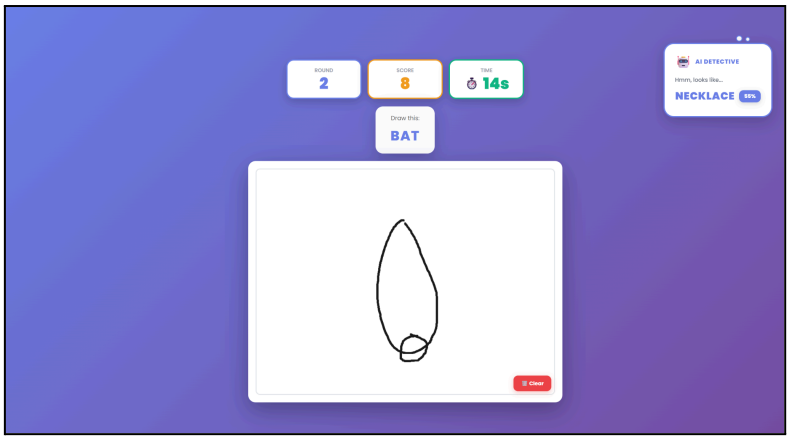


Figure 5.1.2 Deployed App UI

6. Reflection

6.1 Project Strengths

This project demonstrates the effectiveness of a well-regularized CNN for sketch classification, achieving a strong test accuracy of **90.13%** on 130,000 images across 26 classes. The close alignment between training and validation performance indicates that the applied regularization techniques successfully prevented overfitting. The custom architecture, featuring progressive convolutional depth, residual-style connections, and global average pooling, balanced model capacity and generalization while remaining computationally efficient. Data augmentation and mixup further improved robustness to drawing variability.

6.2 Limitations and Future Directions

The primary limitations of this project were **hardware and time constraints**, which restricted batch size, architectural complexity, and the number of classes used from the Quick, Draw! dataset. Only 26 out of 345 available categories were explored due to training cost. Future work may include scaling to more classes, leveraging transfer learning, exploring ensemble methods, and training on more powerful hardware to improve performance and efficiency.

6.3 Conclusion

A CNN-based sketch classification system was successfully developed and trained from scratch, achieving a test accuracy of **90.13%**. The model demonstrated stable convergence and strong generalization, supported by effective regularization and optimization strategies. Overall, the results confirm the feasibility of CNNs for sketch recognition and provide a solid foundation for further improvements and real-world interactive applications.

7. References

1. P. F. Christiano, et al. (2018). *Google's Quick, Draw! Dataset*.
2. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
3. Lorente, Òscar & Riera, Ian & Rana, Aditya. (2021). Image Classification with Classic and Deep Learning Techniques. 10.48550/arXiv.2105.04895.