CS 399: AI

Recreating Google's Quick, Draw!

Ethan Sterling

December 10, 2024

Intro

Quick, Draw!

Why Machine Learning?

- Google-developed online game where players sketch objects or concepts within 20 seconds.
- Uses an AI neural network to guess the subject of the drawing in real time.
- Each user's drawing contributes to the AI's learning, enhancing its prediction capabilities.
- Combines simple gameplay similar to Pictionary with advanced AI technology.

Project Goal: To recreate the Quick, Draw! game using machine learning.

Dataset Overview

- Contains over 50 million hand-drawn sketches contributed by players worldwide.
- Spans 345 categories, showcasing the creative ways individuals represent objects and ideas.
- Sketches are vectorized, storing the stroke-by-stroke sequence to capture temporal and spatial information.
- Selected categories:
    - Airplane, hot air balloon, ice cream, flower, pizza, bicycle, star, camera, dog, radio, envelope, lighthouse, mailbox, and windmill.

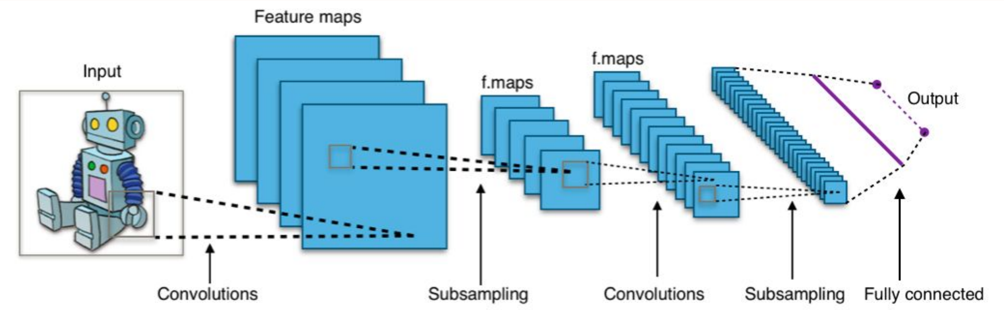
Custom Preprocessing

Tailored for CNN training:

1. Reshaped to (28, 28, 1):
   - Reduced dimensionality for computational efficiency.
   - Converted to grayscale.
2. Normalized Pixel Values:
   - Scaled pixel intensities to [0, 1].
   - Improved model convergence.
3. Label Assignment:
   - Assigned consistent labels based on file order.
4. Class Name Extraction:
   - Extracted human-readable category names from filenames.
5. Data Splitting:
   - Created training (80%) and validation (20%) sets.
   - Mitigated overfitting with separate evaluation data.

Machine Learning Techniques

- CNNs process images while preserving spatial relationships between pixels, critical for understanding sketches.
- Automatically identifies patterns, from simple edges to complex shapes, enabling recognition of diverse sketches.
- Parameter sharing reduces computational costs.
- Handles noise and imperfections in freehand drawings effectively.
- Learns directly from raw pixel data, eliminating the need for handcrafted features and allowing scalability.



Starting Model

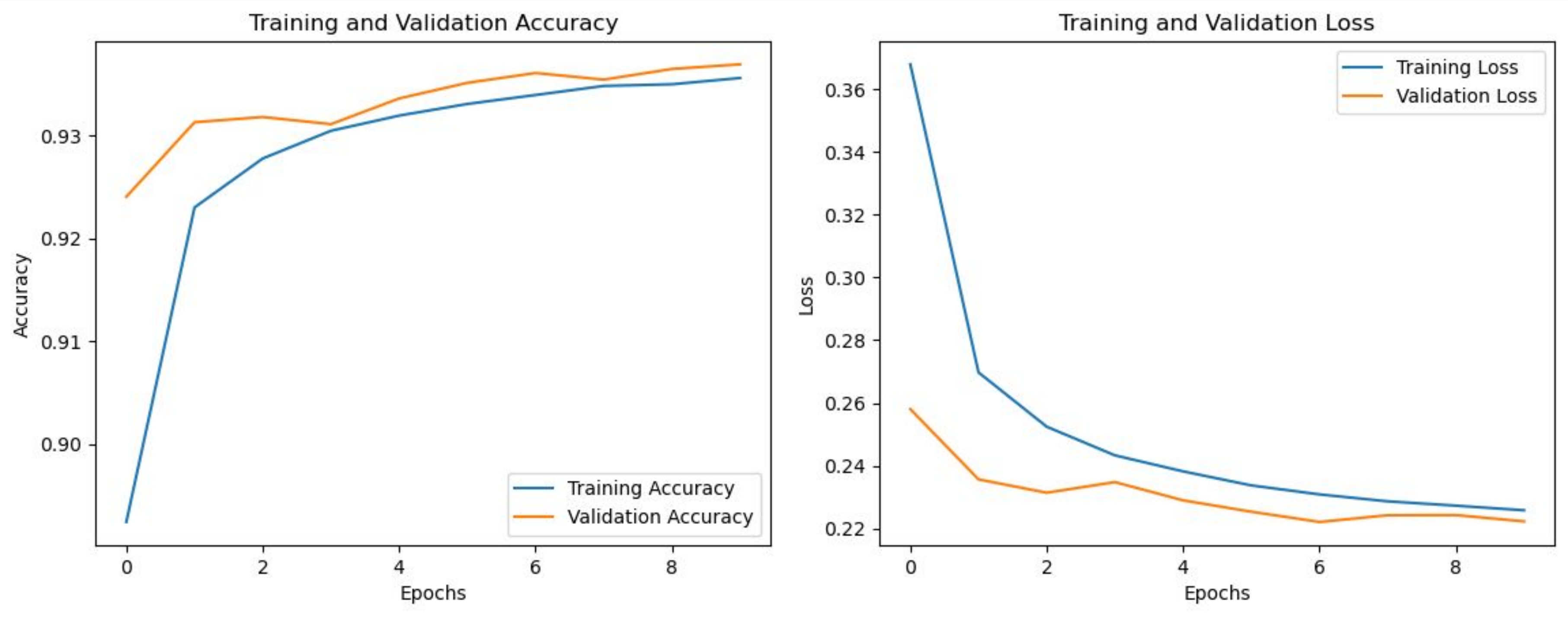
Architecture:

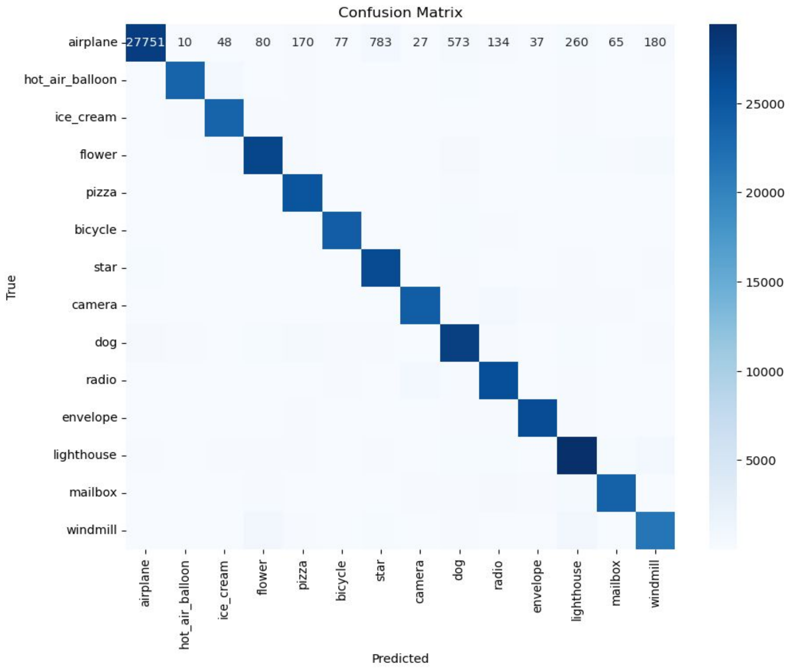
- 3 convolutional layers with small filters (3×3).
- 1 dense layer with 128 neurons.
- Dropout rate: 0.5.

Performance:

- Validation accuracy: 94% (10 epochs).
- Limited ability to capture complex patterns.

```python
# Define the CNN model
def create_cnn_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax'),
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```


Training and Validation Accuracy
Accuracy
Epochs
Training Accuracy
Validation Accuracy
Training and Validation Loss
Loss
Epochs
Training Loss
Validation Loss



Classification Report:

	precision	recall	f1-score	support
airplane	0.95	0.92	0.93	30195
hot_air_balloon	0.97	0.93	0.95	25164
ice_cream	0.95	0.96	0.95	24585
flower	0.94	0.93	0.93	29090
pizza	0.92	0.97	0.95	26042
bicycle	0.97	0.96	0.96	25299
star	0.93	0.96	0.95	27514
camera	0.96	0.94	0.95	25748
dog	0.90	0.91	0.91	30289
radio	0.92	0.94	0.93	27436
envelope	0.98	0.97	0.97	27163
lighthouse	0.89	0.92	0.91	32035
mailbox	0.95	0.92	0.93	25861
windmill	0.92	0.89	0.90	24292
accuracy			0.94	380713
macro avg	0.94	0.94	0.94	380713
weighted avg	0.94	0.94	0.94	380713

Refined Model

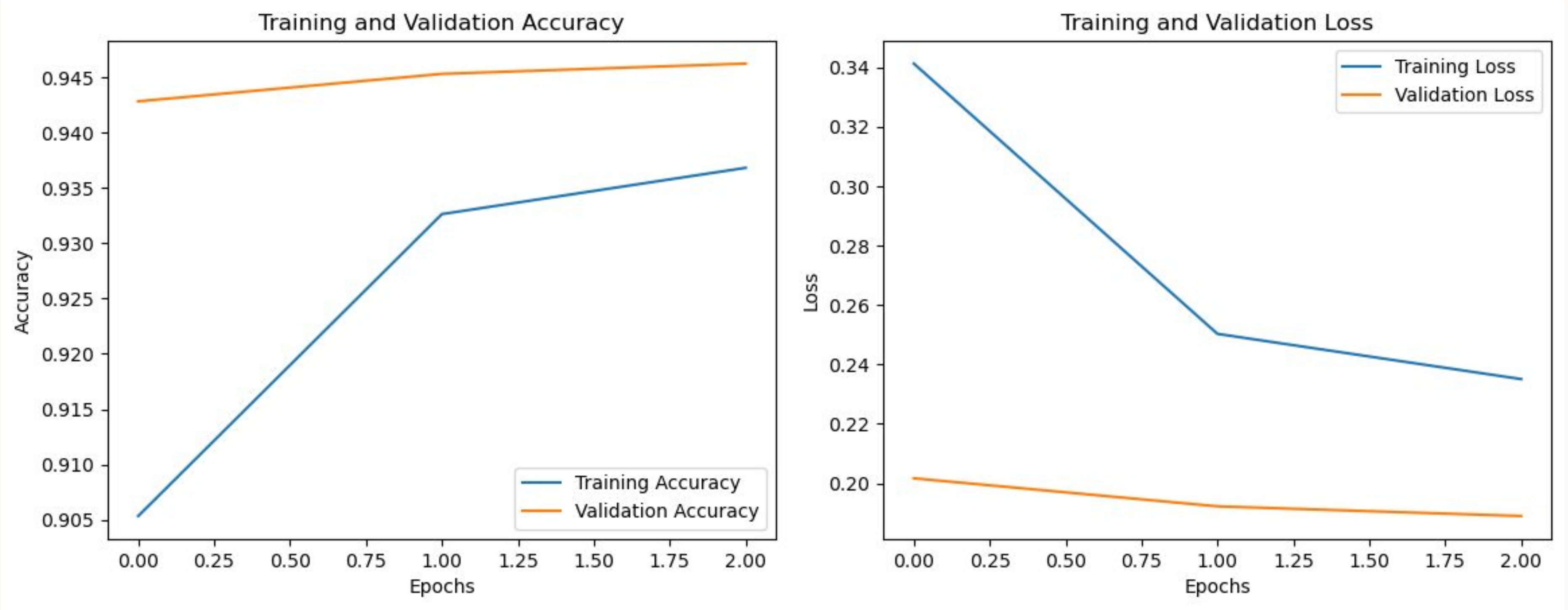
Architecture:

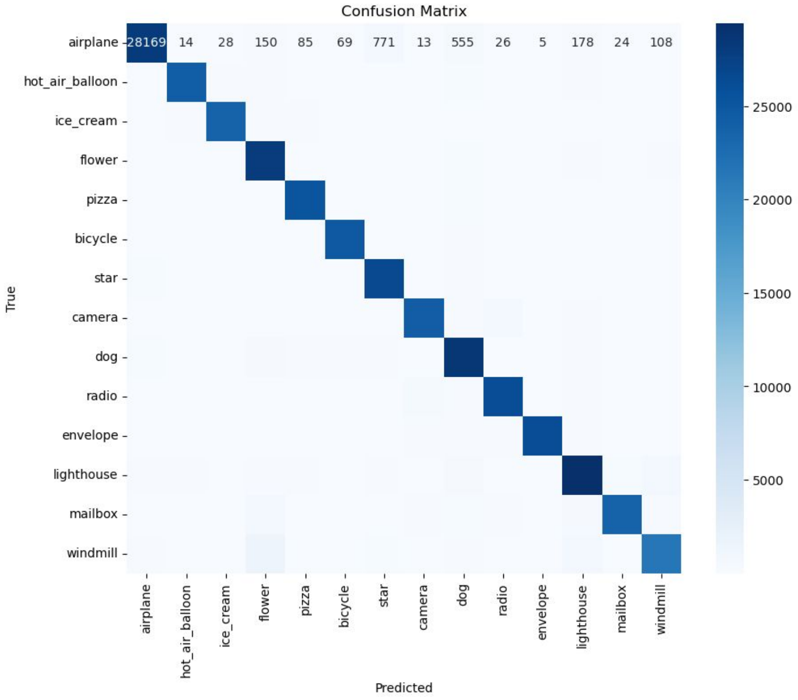
- 2 convolutional layers with larger filters (5×5).
- Dense layers expanded to 512 and 128 neurons.
- Dropout rate increased to 0.6.
- padding='same' to preserve edge features.

```python
# Define the CNN model
def create_cnn_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (5, 5), activation='relu', input_shape=input_shape),
        MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'),
        Conv2D(64, (5, 5), activation='relu'),
        MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.6),
        Dense(128, activation='relu'),
        Dropout(0.6),
        Dense(num_classes, activation='softmax'),
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

Performance:

- Validation accuracy: 95% (3 epochs).
- Faster convergence and improved generalization.


Training and Validation Accuracy
Accuracy
Epochs
0.905
0.910
0.915
0.920
0.925
0.930
0.935
0.940
0.945
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Training Accuracy
Validation Accuracy
Training and Validation Loss
Loss
Epochs
0.20
0.22
0.24
0.26
0.28
0.30
0.32
0.34
Training Loss
Validation Loss



Classification Report:

	precision	recall	f1-score	support
airplane	0.95	0.93	0.94	30195
hot_air_balloon	0.97	0.96	0.97	25164
ice_cream	0.97	0.96	0.97	24585
flower	0.89	0.96	0.93	29090
pizza	0.95	0.97	0.96	26042
bicycle	0.97	0.98	0.98	25299
star	0.94	0.97	0.95	27514
camera	0.96	0.94	0.95	25748
dog	0.90	0.94	0.92	30289
radio	0.95	0.95	0.95	27436
envelope	0.99	0.96	0.98	27163
lighthouse	0.93	0.92	0.92	32035
mailbox	0.97	0.91	0.94	25861
windmill	0.93	0.88	0.90	24292
accuracy			0.95	380713
macro avg	0.95	0.95	0.95	380713
weighted avg	0.95	0.95	0.95	380713

Real-Time Predictions

- Users draw sketches, and the model predicts the category.

Challenges:

- Poor accuracy on live input.
- Likely caused by mismatched preprocessing and noisy input.

Insights:

- Training data needs better alignment with real-world use cases.