AI Lab Report: Hand Gesture Controlled Snake Game Using Deep Learning

Oktay Taha Ates

Abstract

This project presents a gesture-controlled version of the classic Snake game using deep learning and computer vision. Using MediaPipe for hand tracking, TensorFlow for gesture classification, and OpenCV for real-time game visualization, I implemented a model that recognizes hand gestures captured via webcam and translates them into directional inputs. The model demonstrates high accuracy and robust real-time responsiveness. The data processing pipeline, training methodology, the design of the multi-layer perceptron model, and the seamless integration of prediction into gameplay will be further explained in this report.

I. Introduction

Hand gesture recognition is an emerging domain in human-computer interaction, offering an intuitive and hardware-light interface mechanism. This project aims to take advantage of deep learning to classify directional hand gestures (up, down, left, right) for controlling a real-time implementation of the Snake game. Using only a webcam, I built a pipeline capable of detecting gestures using MediaPipe and making predictions through a neural network trained on derived landmark features from hand gestures.

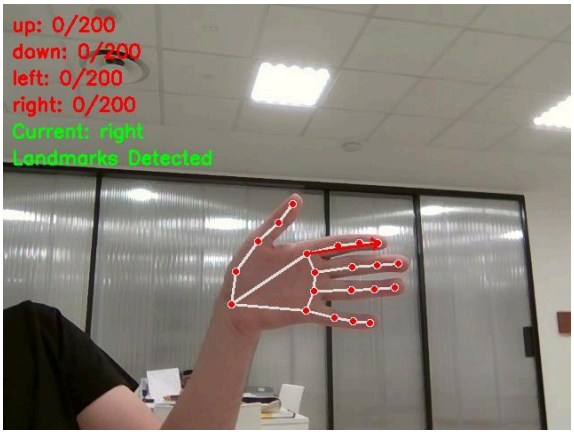
II. Methodology

Mediapipe Hands Overview

MediaPipe Hands is a lightweight and real-time hand tracking solution by Google. It uses a pipeline of palm detection followed by 3D landmark regression. Its ability to deliver consistent and robust landmark predictions under varying lighting and background conditions made it ideal for my project.

A. Dataset Collection and Feature Engineering

Gesture frames were captured using a webcam and categorized into four classes: up, down, left, and right. Categorization of directions was based on the angle of the index finger. A total of 200 frames for each direction was taken. Each frame underwent landmark extraction via **MediaPipe Hands**, which provides 21 hand landmarks (x, y, z coordinates).



To improve model performance and generalization, the raw landmark data was preprocessed into a 72-dimensional feature vector, including:

- Normalized landmark positions (relative to the wrist)
- Index finger angle and its components (sine, cosine)
- Tip-to-wrist vectors
- Average tip positions across fingers (index, middle, ring, pinky)

These features are designed to emphasize gesture-relevant spatial relationships while reducing absolute position sensitivity.

B. Data Augmentation Techniques

To avoid overfitting and enhance model generalization, several augmentation techniques were used:

1. **Gaussian Noise**: Small noise ($\sigma = 0.01$ and 0.02) was added to landmark vectors. This simulates slight variations in hand posture or noise from webcam capture.
2. **Random Scaling**: Vectors were scaled by a factor in the range [0.95, 1.05] to simulate varying hand sizes or distances from the camera.

These augmentations proved highly effective at increasing the diversity of the dataset without needing more raw samples.

C. Model Architecture

A fully connected deep neural network (also known as a multilayer perceptron or MLP) was implemented using the TensorFlow/Keras framework to classify the hand gestures into four directional categories. The design of the model was guided by the need to balance computational efficiency and classification accuracy for real-time use. Below is a detailed breakdown of the architecture and its components:

Input Layer:

- 72 neurons corresponding to the engineered feature vector derived from normalized MediaPipe hand landmarks, finger angles, and spatial relationships. This high-dimensional feature space captures the structural semantics of different hand gestures, allowing the network to learn subtle distinctions.

Hidden Layers:

- **ReLU (Rectified Linear Unit):** Helps the neural network learn complex patterns in the hand gesture data by introducing non-linearity while keeping computations fast. It activates only positive inputs (sets negatives to zero), which makes training more efficient and allows the model to better differentiate between gestures like "up," "down," "left," and "right.". ReLU is especially useful here because it supports real-time performance without adding computational overhead.
- **First Hidden Layer:** 1024 neurons, activated using the ReLU (Rectified Linear Unit) function. This large layer allows the model to begin learning high-capacity feature interactions early in the network. A Dropout layer with a dropout rate of 0.5 follows to prevent overfitting by randomly deactivating 50% of neurons during training.
- **Second Hidden Layer:** 512 neurons, ReLU activation, followed by Dropout(0.4). This layer continues to refine the features learned, reducing dimensionality while retaining meaningful representations.
- **Third Hidden Layer:** 256 neurons with ReLU + Dropout(0.3). Here, the model captures intermediate-level gesture semantics and eliminates co-adaptation among neurons.

- **Fourth Hidden Layer:** 128 neurons, ReLU + Dropout(0.2). This smaller layer performs final refinement and helps compress the feature space before output, enhancing generalization.

Output Layer:

- A Dense layer with 4 neurons, each representing one of the gesture classes (up, down, left, right). A softmax activation function converts the raw logits into class probabilities, ensuring that the sum of outputs is 1.0 .

This hierarchical architecture allows the model to progressively abstract input data, identifying meaningful patterns and relationships associated with each gesture type. Each layer transforms the feature space, increasing the network's capacity to generalize from training data.

Training Strategy:

To train this architecture effectively, I employed several deep learning best practices:

- **Loss Function:** Categorical Crossentropy was used due to the multi-class nature of the classification task. It measures the dissimilarity between predicted probabilities and the true one-hot encoded labels.

- **Optimizer:** The Adam optimizer was chosen for its adaptive learning rate and fast convergence properties. A base learning rate of 0.001 was found to offer a good balance between convergence speed and stability.

- **Callbacks:**

    - **EarlyStopping:** This callback monitors the validation accuracy and halts training if no improvement is observed for 15 consecutive epochs. This helps avoid overfitting and saves computational resources.

    - **ReduceLROnPlateau:** If the validation loss plateaus, the learning rate is reduced by a factor of 0.2, allowing the optimizer to make finer adjustments to the weights.

    - **ModelCheckpoint:** The best-performing model (based on validation accuracy) is saved during training. This ensures that the most generalizable model is preserved.

- **Class Imbalance Handling:**
  To mitigate the effects of unequal class distributions in the dataset, we computed class weights inversely proportional to class frequencies. These weights were passed to the training procedure, allowing the model to treat underrepresented classes more seriously during optimization.

Overall, the architecture and training strategy were designed to ensure strong classification performance while maintaining real-time inference capability and resistance to overfitting.

The model was trained for 50 epochs with a batch size of 32. Validation performance was monitored to avoid overfitting.

D. Game Integration

The trained model was integrated into a real-time version of the Snake game built with OpenCV. A webcam feed continuously supplies frames, from which hand landmarks are extracted using the same preprocessing pipeline. Predictions are made in real time and used to control the direction of the snake.

To ensure responsive and realistic gameplay:

- Prediction latency was kept low (~30 FPS performance)
- Direction changes were filtered to prevent 180-degree turns
- The game grid and camera feed were displayed side-by-side for usability

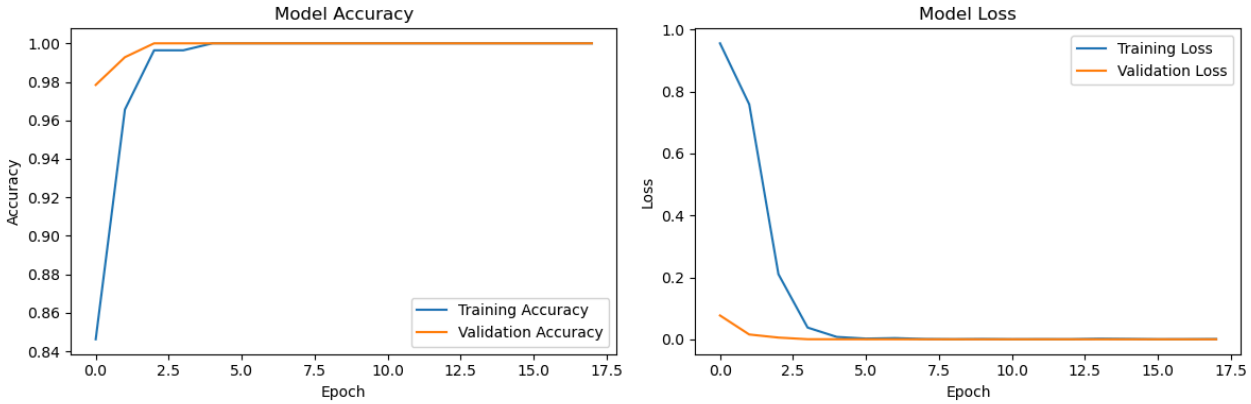
---

III. Results and Discussion

A. Training and Validation Performance

- **Final Training Accuracy**: ~100%
- **Final Validation Accuracy**: ~100%
- **Validation Loss**: Near zero

The model converged quickly, indicating that the extracted features were both rich and robust. The use of Gaussian noise and scaling augmentation proved crucial in achieving near-perfect validation performance.



B. Evaluation Metrics

- Confusion matrix showed perfect classification across all four gesture classes.
- Classification report reflected high precision, recall, and F1-scores for each class.

C. Real-Time Inference Performance

- In-game inference was consistently accurate.
- The frame rate remained smooth (~30 FPS).
- Minor variations in hand orientation or lighting had negligible impact.

D. Challenges Encountered

- **Lighting Conditions**: Lighting changes affected detection stability in some frames.
- **MediaPipe Landmark Enhancement**: The derivation of the 72-dimensional vector caused some trouble because of the normalization and calculations of some landmark points. This was solved by some tweaking in the calculations.

IV. Conclusion

This project demonstrates the feasibility of gesture-based game control using deep learning and webcam input. With minimal hardware and free software libraries, we achieved accurate real-time gesture recognition for directional control. Potential extensions include:

- Prettier UI for the snake game
- Training the model in more diverse environments and lighting conditions
- Implementing more diverse gesture commands to the model
- Implementing a multiplayer gesture-based mode

References

[1] Google MediaPipe Hands - https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker?hl=tr
[2] TensorFlow - https://www.tensorflow.org
[3] Zhang, X., et al. "Hand Gesture Recognition using Deep Learning and MediaPipe.": 2004.10151 (2020)