Real-Time Martial Arts Gesture Classification for Interactive Game Control Systems

Loring Scotty Hoag and Niall Parker

Motivation

Traditional game inputs are keyboard, controllers. But what if we could use our body? This is the problem solved by systems like the Kinect.

Enable gaming for:

- Accessible Control
- Immersive interaction

Challenges:

- Latency
- Hardware Cost
- Hard to Discriminate across variety of human forms

The solution: Machine Learning!!!

FOUR NEW PLAYER CHARACTERS! INTRODUCING

NEW FEATURES!

PLAY AS A BOSS!

TOUGHER ORIGINAL EIGHT!

78 POSSIBLE MATCHES!

How To Play

Insert coin(s) and press 1 PLAYER or 2 PLAYER start button.
Players can choose to be one of 10 different fighters by moving JOYSTICK.
Second player can join in at any time.
Each MATCH consists of best out of 3 rounds. Winner is first player to win 2 rounds.
BONUS ROUND PLAY: TIMING and STRENGTH determine how well you score.

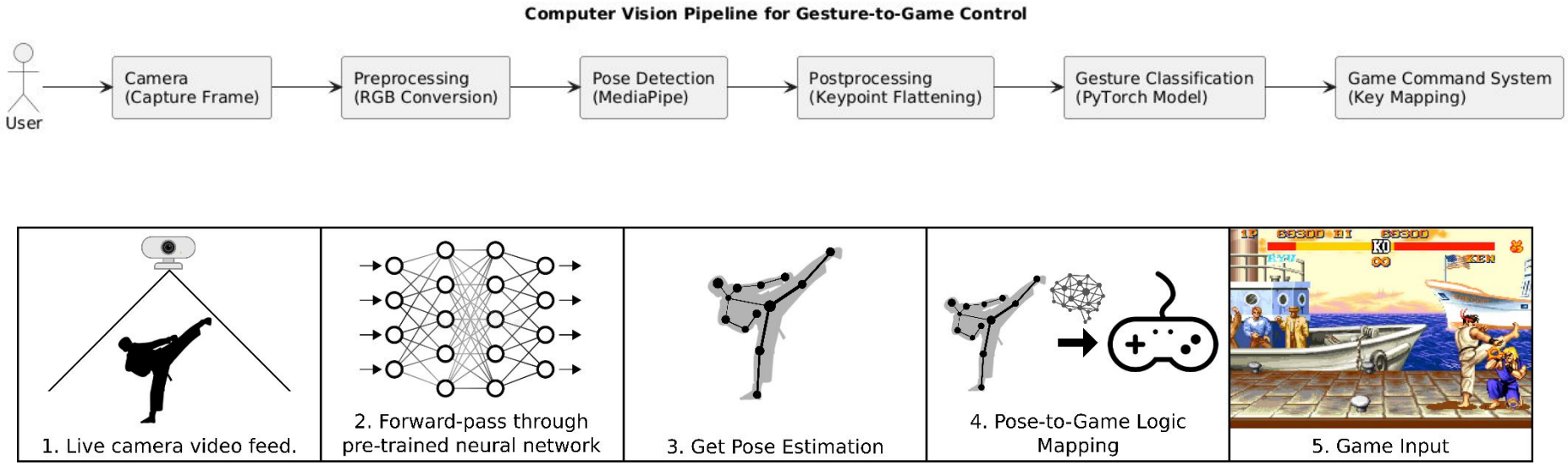
CAPCOM®

Your Controls

Use JOYSTICK and 3 PUNCH buttons to throw a variety of PUNCHES.
Use JOYSTICK and 3 KICK buttons to throw a variety of KICKS.
Each button has a different level of strength and causes damage accordingly.
Pull JOYSTICK back to retreat to DEFENSIVE position.
Discover SECRET TECHNIQUES by various combinations of JOYSTICK and BUTTONS.

THE ULTIMATE FEATURE!

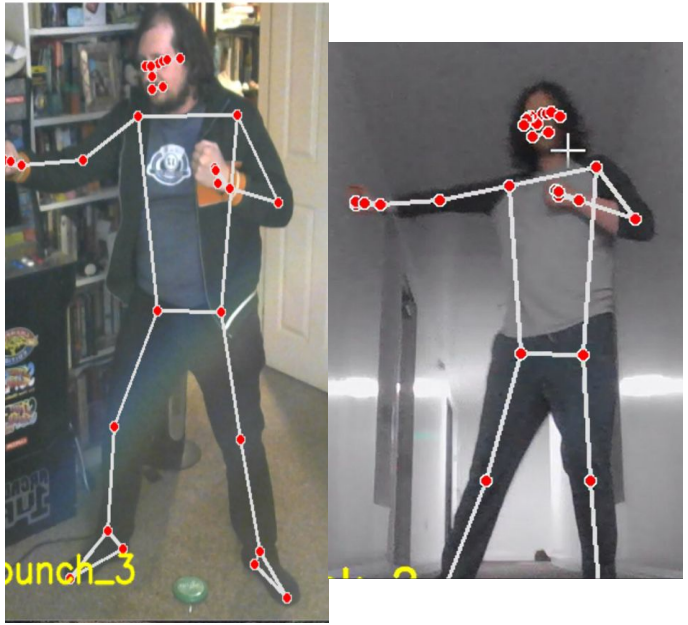
System Architecture



Pose Estimation

For pose detection, we use BlazePose Lite from MediaPipe. It gives us 33 keypoints per frame including joints like elbows, hips, and shoulders. It's fast enough to run in real time on a laptop CPU. We ignore frames where visibility scores are too low — this helps avoid false detections due to blur or occlusion.

- MediaPipe BlazePose Lite
- 33 (x,y,z) keypoints
- ~30 FPS, CPU-friendly
- Filter out low-visibility joints



Gesture Labels

We defined 21 gesture classes tailored to fighting game controls — covering directional movement, basic attacks, and complex moves. Including an explicit "idle" class was important to reduce false positives when the player isn't performing a gesture. Each label maps to one or more key presses.

- 21 total labels
- Movement: jump, left, right, duck
- Attacks: Three types of punches and kicks
- Specials: Left and right variations of Hadouken, Shoryuken, and Tornado Kick
- Throws: One left throw, one right throw
- Other: “Idle” pose, “start”, “select” (for navigating menus)

Dataset Collection

We collected training data by acting out each gesture while recording skeletal landmarks. We ensured class balance and included variation in lighting and clothing to improve generalization. Labels were validated in bulk by comparing similarly labeled images and ensuring that “distance” was close to a mean.

- **~11,100** labeled examples
- Samples per gesture vary.
  Duck: 300-400, Punches: +700 each
  (Similar-looking gestures require more samples to distinguish)
- Varying lighting, color, background

Classifier Architecture

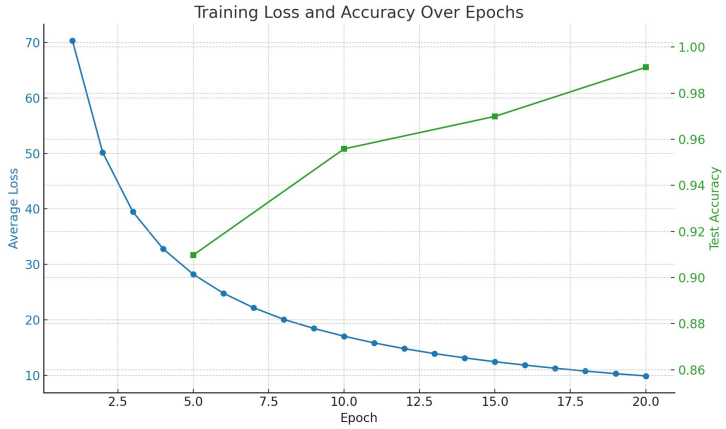
- 99D input (33 joints * x,y,z)
- Three hidden layers (128, 64, 32)
- ReLU + Softmax
- Output: 21 labels

The classifier is a simple three-layer feedforward MLP neural network. The input is a 99-dimensional vector of normalized landmark coordinates. We use ReLU activations in hidden layers and softmax on the output. This structure is fast and lightweight — perfect for real-time performance.

Training Procedure

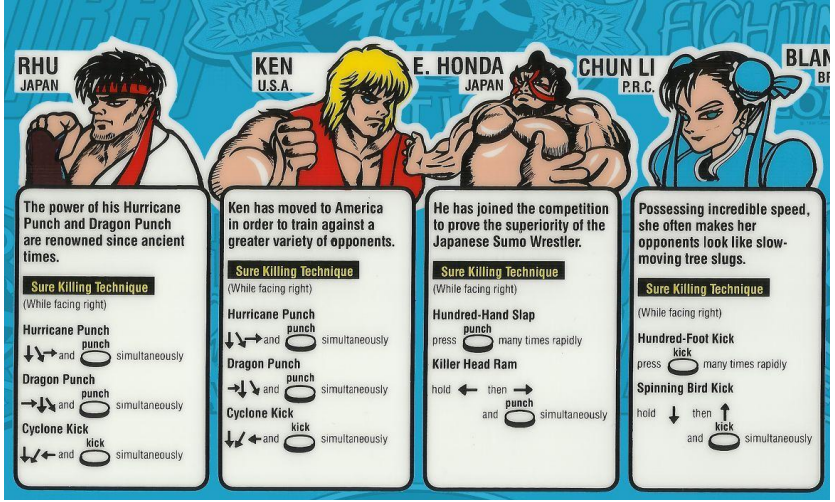
We trained the model using cross-entropy loss and Adam optimizer with early stopping. The dataset was split 80/20 for training and validation. To improve robustness, we added small Gaussian noise to landmarks during training, simulating variation in detection accuracy.

- Early stopping after 50 epochs
- Adam optimizer (lr=0.001)
- Cross entropy loss



Command Mapping

- Single keys: punch → 'n'
- Held keys: move → 'a'
- Sequences: special → ['s', 'd', 'n']
- Injected using Python lib pydirectinput



Once a pose is classified, we map it to in-game actions. Some are simple taps, others are held keys, and some are multi-step sequences. For example, Hadouken is mapped as: [`Down → Down+Forward → Forward → Punch`].

These are sent via the `pydirectinput` Python library to simulate real hardware input for reduced latency required by fast action games. Inputs are processed via a command queue on a separate thread from classification for performance.

Performance Benchmarks

iFighter II Turbo
FPS: 60
Latency: **~30ms,**
(detection to key press)

SNES (Original Console)
FPS: 60
Latency: 33ms - 100ms

SNES Classic
FPS: 60
Latency: 83 - 102 ms

Validation Accuracy: >97%

Win Rate:

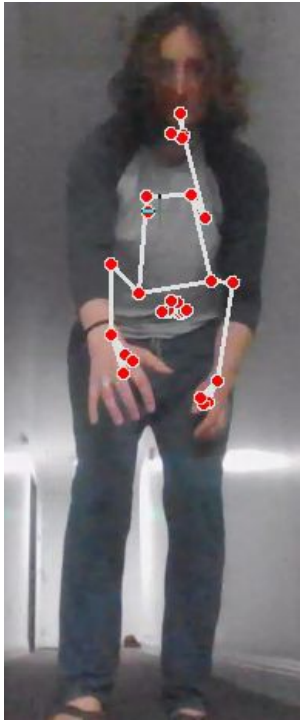
Dependent on Player Skill

Failure Cases

- Occlusion -> Missing Joints
- Fast Motion Blur
- Ambiguous Poses
- Time delay necessary for communicating with emulator can cause missing commands

Fixes: Idle smoothing, pose filtering, state tracking

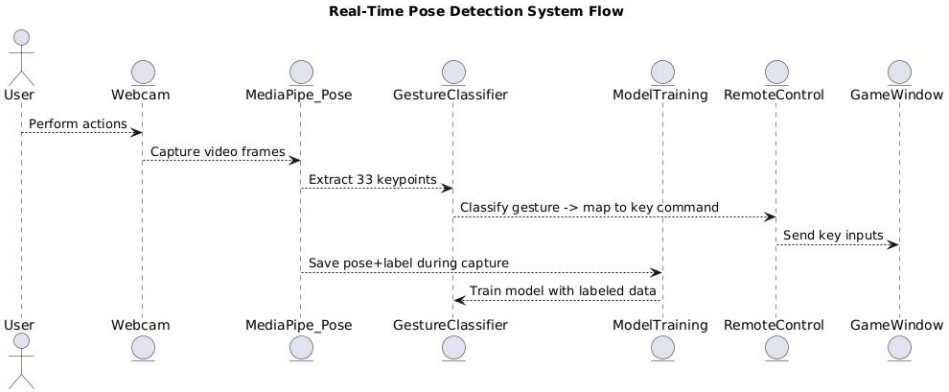
Future Fixes: model transitions, wider dataset,
larger testing space (Requires large area, esp. 2 players)



Real Time System Flow

1. User moves → pose detected
2. Classifier labels frame
3. Label transformed into command sequence
4. Command transmitted to game
5. Game responds in ~2ms (Emulator/Hardware Dependant)

Fully immersive, no noticeable lag



Future Work

- Multiplayer support
- Support for full character roster
- Smoother temporal models (LSTM?)
- VR/AR integration
- Larger datasets
- Integration with more games
- Game input queue rate-limiting
- Better solution for training “jump”

Conclusion

- Full-body gesture control
- Works in real-time on CPU
- Compatible with legacy games
- No special hardware