UMEÅ UNIVERSITY

A Machine Learning Framework for Real-Time Gesture and Skeleton-Based Action Recognition in Unity

Exploring Human-Compute-Interaction in Game Design and Interaction

Arian Moeini

Arian Moeini
Spring 2024
Degree Project in Computing Science and Engineering, 30 credits
Supervisor: Zoe Falomir
Extern Supervisor: Hannes Paulsson
Examiner: Henrik Björklund
Master of Science Programme in Computing Science and Engineering, 300 credits

Abstract

This master thesis presents a machine learning framework for real-time gesture and skeleton-based action recognition, integrated with the Unity game engine. The system aims to enhance human-computer interaction (HCI) in gaming and 3D related applications through natural movement recognition, by training a model on skeleton tracking data. The framework is trained to accurately categorize and identify gestures such as kicks and punches, enabling a more immersive gaming experience not existing in traditional controllers.

After studying the evolution of HCI and how machine learning has transformed and reshaped the interaction paradigm, the prototype system is built through data collection, augmenting, and preprocessing, followed by training and evaluating a Long Short-Term Memory (LSTM) neural network model for gesture classification. The model is integrated into Unity via Unity Sentis using Open Neural Network Exchange (ONNX) format, enabling efficient real-time action recognition in 3D space. Each component of the pipeline is available and adaptable for future customization and needs, skeleton tracking and Unity integration is built using the ZED 2i camera and ZED SDK.

Experimental results demonstrate that the system presented can achieve over 90% accuracy in identifying predefined gestures. As a bridging solution tailored for Unity, this framework offers a practical solution to action recognition that could be found useful in future applications. This work contributes to advancing human-computer interaction and offers a foundation for further development in gesture-based Unity game design.

Acknowledgements

I would like to express my gratitude to Umeå University for providing the academic environment and resources that made this work possible. My sincere appreciation goes to my supervisor, Zoe Falomir, for her guidance and unwavering support throughout the research process.

A heartfelt thank you also goes to Hannes Paulsson, my external supervisor, for allowing me to conduct my master's thesis work at his company.

Lastly, I would like to extend my gratitude to all those who contributed directly or indirectly to the success of this project. Your encouragement, feedback, and assistance were invaluable.

Contents

1 Introduction

In the fast-evolving field of machine learning and human-computer interaction (HCI), the mission of integrating new and evolving interaction paradigms has gained new attention. The need for natural and intuitive interfaces has been a must in all regards of HCI and machine learning is opening up new revolutionizing solutions. One of the territories with immense potential has been action based integration without the need for embodied digital artifacts. This master thesis explores the intersection of real-time input streamed video, machine learning, action recognition and HCI. By implementing an architecture that enhances users' interactive gaming experience, through training machine learning models based on skeleton tracking data.

1.1 Background

As digital environments have become much more integrated into our daily lives, the emergence of virtual reality (VR), augmented reality (AR), and mixed reality (MR) platforms gained popularity in the recent decade. New technology is adapting these paradigms using sensor technology to gain relevant data on positional tracking in 2D and 3D space. In this context, machine learning has emerged as a pivotal technology with its accessibility and affordability. The usage of these models gives an accurate interpretation of the sensor input, which could potentially create reliable systems. A specific field of interest when it comes to this technology is interactive gaming and how the systems can benefit from these technological advancements to create immersive experiences. Moreover, a study by Bowling et al. (2006), explore the realm of machine learning in computer games, establishing its significant role in enhancing game intelligence and player engagement.

1.2 Motivation

The work carried out in this master thesis is driven by the vision of unlocking the physical barrier between the player and the digital world. Traditional gaming interfaces rely mainly on buttons and controllers to gain physical engagement from the player. One early step towards breaking this barrier was Microsoft's Kinect, which enabled controller-free gaming by using depth sensors and computer vision to recognize and interpret users' gestures. This interaction paradigm has brought new possibilities for interactive gaming by allowing players to use their whole body, creating an immersive and activate engagement with the player. While Kinect was groundbreaking, it had limitations in terms of precision and predicting users' movements directly in space and time. Nowadays, we can envision a future which includes more sophisticated interaction mechanisms driven by machine learning and action recognition, in which the user could be interacting with less constraints and more freely. That is by not having a bridging controller system that translates the wanted actions but instead creating a solution that can be reactive to a sequence of actions in real-time.

1.3 Research Objectives

The research objectives pursued in this master thesis are the following:

1. Develop a system that allows the user to interact with a game:
   - (i) That allows real-time interaction with the Unity game engine using gesture and skeleton-based action recognition.
   - (ii) Enhance user experience by enabling intuitive game control through natural movements.
2. Create a Robust Machine Learning Model:
   - (i) Design and train a Long Short-Term Memory (LSTM) neural network to accurately identify and categorize user gestures such as kicks and punches.
   - (ii) Optimize the model for real-time performance with minimal latency, ensuring responsiveness suitable for gaming.
3. Integration and Application:
   - (i) Seamlessly integrate the trained model into the Unity engine using the ONNX format, enabling real-time action recognition in 3D space.
   - (ii) Demonstrate the practical application of this system within a prototype game, showcasing its effectiveness and the enhanced interaction it supports.
4. Evaluate System Performance: Construct an evaluation framework to assess the accuracy and efficiency of the gesture recognition system.

1.4 Master Thesis Structure

This master thesis is organized as follows:

- **Section 1** introduces a background to the field of machine learning and human-computer interaction (HCI).
- **Section 2** presents a comprehensive review Of Existing Literature, which includes an in-depth review of the evolution of HCI and its interaction paradigms, with respect to machine learning and how it has changed the landscape.
- **Section 3** describes the methods used for the technological advancement developing a prototype game capable of efficient and accurate action recognition in real time, utilizing skeleton tracking as its core technology.
- **Section 4** explores the application and Integration, that is, how to efficiently integrate the system with the Unity game engine, giving a demonstration in the potential of bridging the system into interactive gaming experiences.

- **Section 5** reports the evaluation and analysis, where we present a evaluation framework to score the model’s accuracy, efficiency, and impact on user experience.

2 Literature Review

This section presents the trends and challenges of Human-Computer Interaction (Section 2.1). The integration of Machine Learning within HCI (Section 2.2) and into various action recognition techniques (Section 2.3)

2.1 Human-Computer Interaction (HCI): Trends and Challenges

Due to the rapidly evolving technological advancements in the field of Human-Computer-Interaction (HCI), a multidisciplinary area focused on the design, development, and evaluation of interactive computing systems for human use, many branches are being introduced and actively discussed among researchers. The contributions by Rashid et al. (2023); Dhakecha (2022); Victorelli et al. (2020) provide insightful perspectives into the latest trends and challenges within HCI and are reported next.

Rashid et al. (2023) analyzes the ethical considerations, user-centered design practices, and the sociocultural implications of interactive systems. These are the challenges that HCI designs currently face in all aspects, including machine learning. Creating interfaces that not only are efficient, but also ethically sound and culturally sensitive is crucial for sustainable architectures, that strive for progress while not being detrimental to certain sectors of society. This is well presented by recent developments in Large Language Models (LLMs), where critiques have emerged regarding the biases in outputs linked to unbalanced datasets.

Dhakecha (2022) presents a methodological study on the current state-of-the-art HCI in technologies such as cloud computing, augmented reality (AR), and virtual reality (VR). Dhakecha promotes a fusion of human-centered design with agile interface design methodologies for future HCI interfaces. They envision that HCI will be a key area of research within the AI community for the future, as HCI will not rely as much on physical technologies.

Lastly, Victorelli et al. (2020) explore HCI in the realm of data, calling it Human-Data Interaction (HDI). An area of interest in combination with Artificial Intelligence that seeks to effectively create architectures that help people interact with the huge amounts of data in the modern world. The paper sheds light on the problems related to analyzing and interacting with large datasets, and how designing systems with HDI in consideration is crucial for enhancing HCI.

2.2 Machine Learning in HCI

Machine learning has become a fundamental resource for creating new innovative interaction paradigms. As for the latest years, multiple new ways of using AI systems have been implemented in different aspects of society. In an article written by Rahul Rai & Dolgui (2021). They talk about how machine learning has impacted the manufacturing industry, enabling intelligent support systems that effectively simplify manufacturing tasks despite challenges such as big data management and cybersecurity.

Another publication by Fatima et al. (2023) discusses the impact of ML has had on healthcare, highlighting multiple key sections as data management, clinical operations, drug research, and surgery. The arguments are further addressed in an article by Vanshika & Gupta (2022) where they conclude that neural networks and support vector algorithms could outperform pattern recognition in medicine, showcasing its benefits in transforming the industry.

New technology such as Large Language Models (LMM) has been rapidly increasing in usage. OpenAI's ChatGPT reportedly gained over 1 million users within the first few days of its launch and reached 100 million users in the first 2 months. This achievement marks it as the fastest-growing consumer application to date (Homolak, 2023).

2.3 Action Recognition Techniques

Traditional Methods

Machine learning has developed significantly in action recognition. However, before this widespread incorporation of large language models (LLMs), traditional methods primarily relied on handcrafted features and classic computer vision techniques. Approaches like optical flow and silhouette analysis were the state of the art but they faced many limitations. In a paper written by Moussa et al. (2015), they discussed SIFT (Scale-Invariant Feature Transform), which is an interest point detection technique originally patented by Lowe (1999). Which gave high accuracy scores on the KTH and Weizmann datasets and showed to be an effective approach before the dominance of deep learning models. Another article by Goudelis et al. (2013) explored the potential of the "Trace" transform in recognizing human actions and introduced two novel methods for feature extraction. The methods also showed competitive results on the KTH[1] and Weizmann[2] action datasets.

These studies provide an overview of traditional methods used before the current era of machine learning. While some models relied on manual feature extraction, which could make solutions computationally expensive and less generalizable across different actions and environments, others employed automatic but not learned techniques. Newer solutions, such as deep learning and advanced machine learning techniques, can automatically learn and adapt to complex features (including noise) from raw data, offering a more reliable and adaptable approach.

Machine Learning Approaches

The rise of these systems has led to advances in methodologies and improved the accuracy of recognition systems. Nguyen et al. (2022) introduced techniques to improve the generalization of the underlying characteristic. Their work highlights the challenges that come with over-fitting when working with small dataset sizes. They solved the problem with semi-weak supervised learning on unlabeled data, presenting the importance of feature

[1] `https://www.csc.kth.se/cvap/actions/`

[2] `https://www.wisdom.weizmann.ac.il/~vision/SpaceTimeActions.html`

generalization in action recognition.

Another interesting article was written by Gaikwad et al. (2023). Their work analyzes an action recognition system to help doctors monitor patients. The system was made adaptable to other areas, such as banking security systems. The system was made to classify human activity, resulting in an architecture type that was optimal for the project.

In another paper, Gill et al. (2023) tackle the challenge of human action detection through an innovative approach utilizing the EfficientNetB3 model. This article discussed the shift in going from action recognition to action prediction, which involves forecasting the next states given the current state. For now, most of the systems are built to recognize actions that have been fully executed. With the growing interest in this area, creating systems that can infer the next potential state would be significant progress in this field.

Skeleton Tracking Technologies

Skeleton tracking continues to develop and expand as a paradigm within machine learning and human-computer interaction (HCI). The approach of building Long Short-Term Memory (LSTM) network and integrating them with skeleton data has notably enhanced action recognition, giving a tool that can adapt and learn from sequences of data. Recent research by Yuna et al. (2022) and colleagues at Akita University has used LSTM models to create a model that recognizes human actions. Their method, built on raw skeleton data and the models ability to infer missing joint data and compute dynamic features from the skeleton achieved an average recognition accuracy of 93.1%.

Beyond LSTM, other sophisticated methods have gained prominence. The recent studies published by Cheng et al. (2023) used Graph convolutional networks (GCN) for handling the spatial-temporal aspects of skeleton data, showing significant accuracy boosts in environments with cluttered backgrounds and varying illumination. These methods, in combination with conventional CNN and RNN approaches, give a solution for efficiently capturing the complex dependencies among skeleton joints.

3 The Prototype/ Method

This chapter presents an overview of the system (Section 3.1) and the data collection procedure for training and evaluating the system (Section 3.2). thereafter, data preprocessing and feature extraction (Section 3.3), model training and fine-tuning (Section 3.4), and finally, model deployment and integration with the Unity engine for real-time action recognition (Section 3.5).

3.1 System Overview

The core of this work revolves around a pipeline of integrated systems, giving an adaptable workflow from data collection and preprocessing, to model training and fine tuning and lastly deployment within a gaming environment, specifically for Unity engine integration. The system is divided into different blocks, each responsible for handling a unique task within the action recognition process. It is developed using skeleton tracking as its core tracking mechanism and machine learning for processing. The system architecture is designed with flexibility in mind, giving the potential to change any part of the project to meet the requirements for a certain problem. It includes real-time data processing capabilities with minimal latency and high accuracy levels. The steps involved in this system, as illustrated in Figure 1, are detailed as follows:

1. **Data Collection**: Collection of initial data to build an action recognition database.
2. **Data Augmentation**: Applying transformations to the data to expand the dataset and make it ready for training.
3. **Model Training**: Training the model to classify the different actions.
4. **Model Fine-tuning**: Further refinement of the model to improve performance based on evaluation metrics.
5. **Model Testing and Exporting**: Testing the model to ensure desired accuracy followed by exporting and pipe lining it to Unity.
6. **Unity Integration**: Importing the model into Unity for real-time action recognition within the gaming environment.
7. **Real-Time Action Recognition Converted to 3D Space**: Recognition of user actions in real time and converting them to appropriate 3D representations in the game environment.

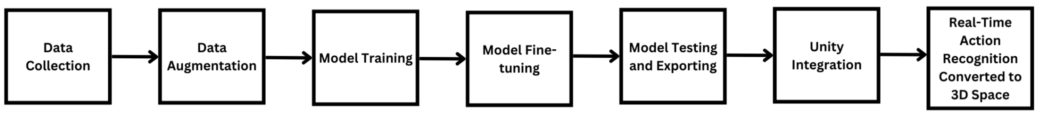


Figure 1: A overview of the different components and sections of the system.

3.2 Data Collection

The data collection begins by establishing directories for the different actions we will try to detect. These paths are used to store the exported numpy array data. We then create 30 sequences, each sequence being 30 seconds long. This approach is derived from the frame rate at which the collection is recorded for this project. Training and recording were done at 30 frames per second for this project. This results in a data amount of 30 actions spanning 1 second in duration.

3.2.1 Utilizing the ZED Camera SDK for Data Acquisition

Data is acquired through the utilization of 2D skeleton data captured via the ZED2i camera's Software Development Kit (SDK)[1]. The camera features various sensing capabilities, including body tracking. The module focuses on bone detection and tracking, representing each detected bone as keypoints that can be output as a 3D or 2D position. It also uses a trained neural network for the detection combined with depth and positional tracking, giving support for body tracking of multiple individuals simultaneously.

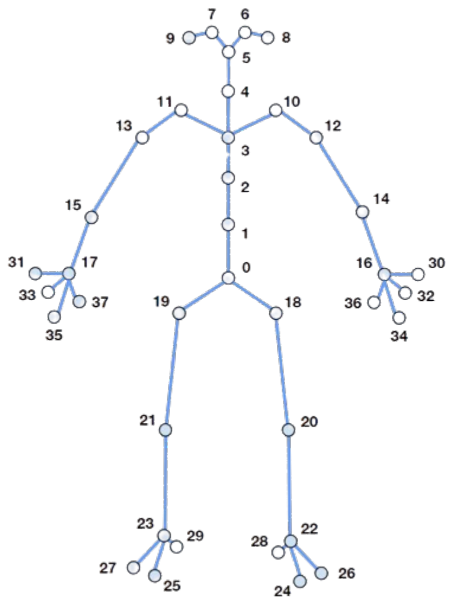


Figure 2: The camera body format contains 38 keypoints following this configuration.

[1]https://github.com/stereolabs/zed-sdk

As Figure 2 shows, the implementation utilises 38 keypoints indexed between 0-37 and extracts the stored positional 2D keypoints as a vector of [x,y]. The configuration parameters are shown in Table 1.

Table 1 Configuration parameters for the ZED2i camera.
Note: *For outdoor scenes or long-range, the confidence should be lowered to avoid missing detections. For indoor scenes or closer range, a higher confidence (∼20-30) limits the risk of false positives and increases precision (∼50+).*

Parameter	Value
Video mode	HD720
Output Resolution	2560x720
Frame rate	30 FPS
Depth mode	PERFORMANCE (fastest)
Body Tracking Model	HUMAN BODY FAST
Detection Confidence Threshold	40

In conjunction with the Stereolabs ZED SDK and OpenCV[2] , the methodology entails iterating over the sequences to capture video frames. During this process, the first identified body within these frame, is stored as a struct containing the body's keypoints. Key points are subsequently exported to both a CSV file and a numpy array[3] for further analysis.

3.2.2 Preprocessing Data and Creating Labels and Features

For preprocessing the data and creating our training and validation set packages the sklearn[4] and Keras[5] was utilized. The prepocessing routine will be based upon sequence learning using time-series data.

The sequences are converted into a numpy array and reshaped so that each sequence is a flat array of features. Action labels are one-hot encoded, converting them into binary class matrices (with length equal to the number of classes) that are appropriate for classification tasks with more than two classes.

This preprocessing results in:

- X shape of $(90, 30, 76)$, representing $(\text{Sequences}, \text{Frames}, \text{Keypoints per Frame})$.
- y shape of $(90, 3)$, representing $(\text{Sequences}, \text{Actions})$.

The dataset is then divided into training and test sets, with 5% of the data reserved for testing. This split is crucial for evaluating the model's performance on unseen data.

Table 2 shows the specific shapes of the training and test datasets following the split. The training set (X_{train} and y_{train}) consists of 95% of the original data, while the test set (X_{test}

[2]https://opencv.org/
[3]https://numpy.org/
[4]https://scikit-learn.org/stable/
[5]https://keras.io/

and y_{test}) is formed using the remaining 5%. Each sample in the feature set (X) contains 30 frames with 76 feature dimensions, representing skeletal points for a 1 second (30 FPS) sequence of action data. The label sets (y) contain the corresponding target outputs for each sample, which in this case are the 3 actions kick, boxing and notFighting that the model will be trained on.

Table 2 Shapes of the datasets after splitting into training and testing sets.

Dataset	Shape	Description
X_{test}	$(5,30,76)$	Test set features
X_{train}	$(85,30,76)$	Training set features
y_{test}	$(5,3)$	Test set labels
y_{train}	$(85,3)$	Training set labels

4 Model Development

This section presents and overview of the model architecture (Section 4.1) and the selection and configuration of the optimization algorithm in model training (Section 4.2). It also addresses the implementation of early stopping and model fitting (Section 4.3), the results of the model training phase (Section 4.4), model evaluation and testing before Unity integration (Section 4.5). And lastly, the process of converting the model for Unity integration (Section 4.6).

As our data being sequences of video frames (time-series data), we build the model architecture to be sequential with recurrent LSTM layers. This gives a structure that works well with temporal dependencies within the data points. In action recognition, the movement of a person in a frame is related to the movement in the next, this sequence of events is crucial to define the correct action being preformed. Sequential models also provides a support for changing the simplicity or complexity as needed by simply adding more layers, giving flexibility to adapt the architecture to a diverse data input. Figure 3 shows the machine learning model architecture which consist of:

- **LSTM Layer:** The layer is a 64-unit LSTM layer. Due to the input data being analyzed sequentially, starting with this layer is essential to recognize patterns and capture both short-term and long-term dependencies. After the layer there is a tanh activation function to squash the output values into a range between -1 and 1.
- **Dropout Layer:** A dropout layer with a rate of 0.5 is set to randomly change a fraction of the input units to 0 at each update during training time to prevent overfitting.
- **Dense Layer:** A dense fully connected neural network layer that maps the output from the LSTM layer to a new space.
- **Second Dense Layer** A last dense layer with the same amount of units as the output actions. As its output later, the activation function is softmax to give a probability distribution for multiclass classification.

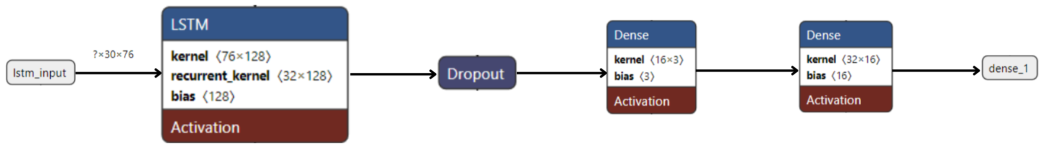


Figure 3: Model architecture.

4.1 Optimization Algorithm Selection and Configuration in Model Training

One of the main components of the training process is the optimization algorithm, which in its core iteratively updates the model parameters to minimize the loss function. After evaluating the options, the **Adam optimizer** was selected as the loss function algorithm, mainly for its robustness and efficiency, but also for being a common choice in complex action recognition tasks.

Adam, an acronym for Adaptive Moment Estimation, combines attributes from two other extensions of stochastic gradient descent: Adaptive Gradient Algorithm (**AdaGrad**) and Root Mean Square Propagation (**RMSProp**). The algorithm employs an approach where it computes the exponential moving average for both the gradient and its squared form. The decay rates of these averages are governed by β_1 and β_2. β_1 of the moving averages of the gradients and β_2 of the moving averages of the squared gradients.

The learning rate is an essential hyperparameter that impacts the speed at which the network learns. To gain an optimal training process, the parameter need to be set at a rate that leads to more precise convergence; this will allow the optimizing function to navigate the gradient landscape without overshooting or getting stuck in local minimals. A learning rate of 0.0007 was set by experimentation, yielding a balance between training speed and stability of performance on the dataset.

Recurrent neural network architectures, such as LSTM have a well-known history of problems with exploding gradients, due to the cumulative effect of gradients over time. To address this, a gradient clipper `clipnorm` was set at 1.0. Any gradients averaging over the set threshold will be scaled down proportionally, stopping any single update to disproportionately impact the weights of the model.

For this multi-class classification task, the loss function chosen was `categorical crossentropy`. And to judge the performance of the gesture recognition model, `categorical_accuracy` was chosen as a metric. Giving a metric that computes the amount of correctly predicted classes to true classes, giving a insightful quantification of the training's efficacy.

These parameters and metrics give a structured foundation for model training and evaluation, enabling the advancement into the experimental stage of the research.

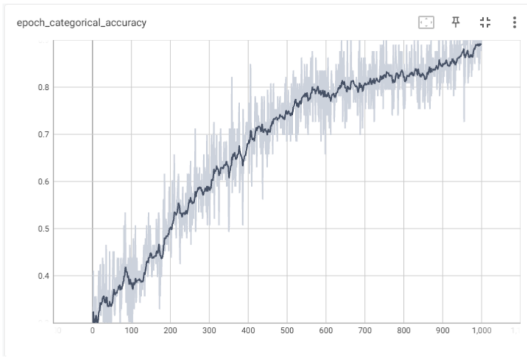
4.2 Implementing Early Stopping and Model Fitting

To polish the training process and prevent overfitting an **Early Stopping** mechanism was integrated using TensorFlow's Keras callbacks. Overfitting takes place when the models learns the training data well, resulting in it capturing noise and small details that would negatively impact its performance on unseen data. This problem is then solved by using regularization to avoid the training process to continue when the models performance on the validation set stops improving over a set amount of epochs.

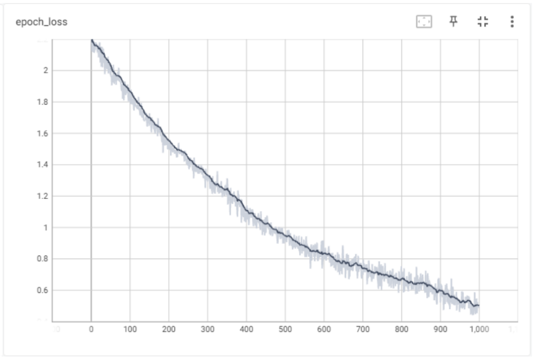
The callback monitors the validation loss (`val_loss`), which represents the error levels on the validation set during training. The `patience` parameter represents the amount of epochs to wait after the minimum loss has been observed before stopping the training. A small portion (10%) of the training data is positioned for validation purposes. This is essential to test the models performance on unseen data during training.

4.3 Model Training Results

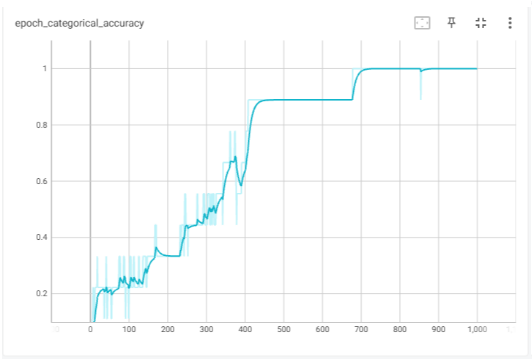
Figure 4 shows the results of the model training phase, as visualized through TensorBoard. Two key metrics were monitored: epoch_categorical_accuracy and epoch_loss for both the training and validation datasets over the course of 1000 epochs.



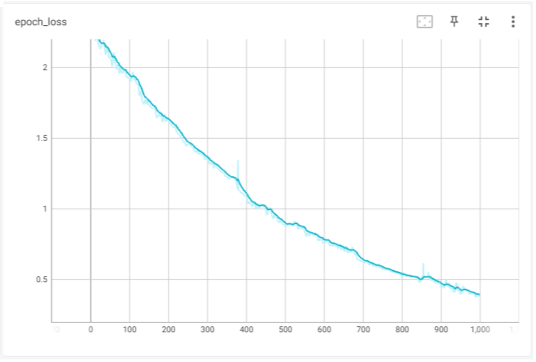
(a) Training Categorical Accuracy



(b) Training Loss Function



(c) Validation Categorical Accuracy



(d) Validation Loss Function

Figure 4: The results of the model training phase, as visualized through TensorBoard.

The **categorical accuracy** is a measure of how well the model has precisely predicted the action that was presented. The results plotted in Figures 4a and Figure 4c presents a consistent upward trend, indicating that the models ability to correctly predict the gesture categories is increasing with the epochs progress.

As seen in the results, the training set is consistently increasing linearly. While the validation accuracy is aswell, it does so with noticeable volatility. This is typical for the validation set due to the model being exposed to new unseen data. The continued increase in both the training and validation accuracy suggests that the model is learning general patterns and not overfitting on the training data.

The **loss function** graph gives a representation of the model error or deviation from the true data labels. The given illustrations of Figure 4b and Figure 4d present the results over time, both training and validation loss decrease significantly over time, converging towards lower values. This is a favorable outcome, suggesting that the model is minimizing the errors in its predictions. Both graphs follow a downward trend that is parallel to each other as seen in Figure 5, indicating that the model is improving and not overfitting.

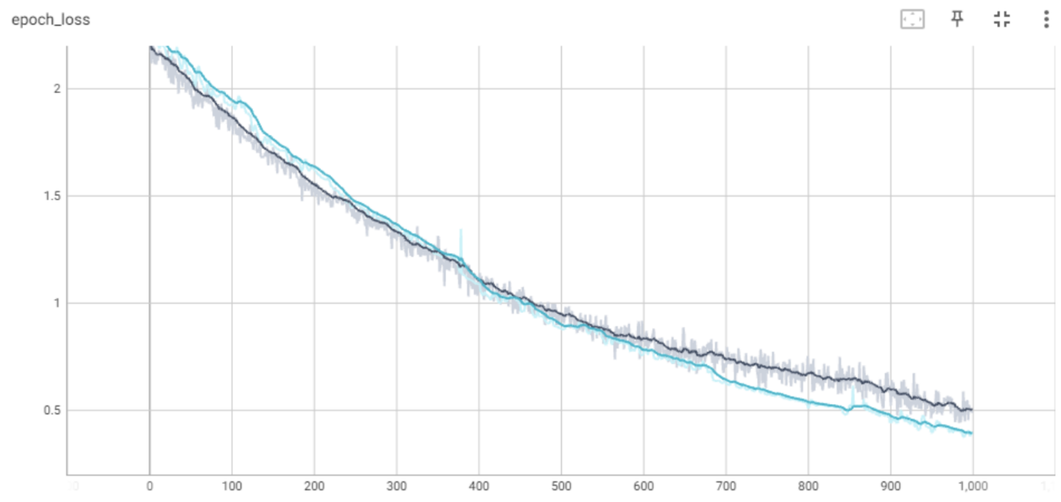


Figure 5: The training loss (dark blue line) and validation loss (light blue line) parallel to each other

The smoothing parameter in TensorBoard was adjusted to 0.9, providing a clearer view of general trends by reducing the noise in the plots. This gives us a smooth overview that makes it easier to identify the overall pattern without focusing on short-term fluctuations that are inevitable in the stochastic nature of the training process. The results demonstrated by these graphs present a successful training phase with an appropriate choice of hyperparameters and regularization strategies.

4.4 Model Evaluation and Testing before Unity Integration

The performance of the model is summarized by the following metrics.
The efficacy of the model is summarized by its **accuracy** and **F1 score**, which are crucial metrics in evaluating predictive performance for classification tasks. The model achieved an accuracy of **1.0** and an F1 score of **0.903** on the test set.

The high accuracy indicates a strong overall classification performance. However, achieving 100% accuracy might indicate issues with the training that are not presented. Therefore, the effectiveness of the model must be verified in practical applications to determine if the metrics might be misleading. The F1 score, which is the harmonic mean of precision and recall indicates a balanced detection capability for both positive and negative classes. Both these scores portrays the model's robustness in classifying tasks.

Figure 6 represents the confusion matrices given by the performance of the model during the evaluation. Each confusion matrix summarizes the classification results for the X_test set.

Each confusion matrix includes the following components:

- **Axes:**
  - The *vertical axis* (y-axis) represents the true labels in the test data.
  - The *horizontal axis* (x-axis) represents the predicted class by the model.
- **Cells:**
  - Each cell presents the number of classifications that corresponds to the predicted class.
  - *Diagonal cells*, running from top-left to bottom-right, indicate correct classifications where the predicted label matches the true label.
  - *Off-diagonal cells* represent instances where the model's predictions did not match the true labels (misclassifications).
- **Color Scale:**
  - The intensity of the color in each cell corresponds to the count of samples classified into that category, with brighter colors meaning higher counts.

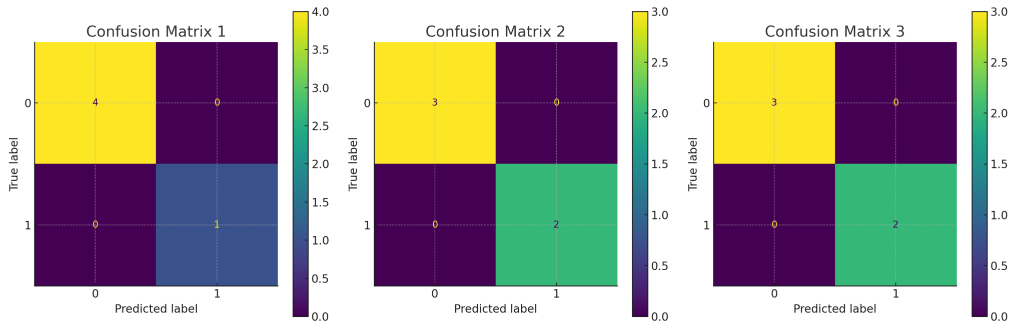


Figure 6: Confusion Matrix 1 = Boxing, Confusion Matrix 2 = notFighting, and Confusion Matrix 3 = Kick. Confusion matrices summarizes the classification results for the X_test set.

The results show a perfect accuracy over the five examples given. This shows a promising but potentially misleading outcome that needs to be further investigated.

Real-Time Testing and Model Deployment

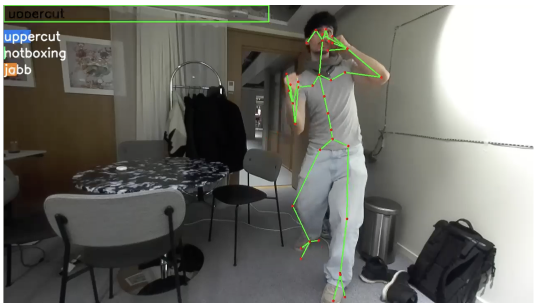
After reaching a prediction accuracy of over 90% on the test set, the next phase involves real-time testing to validate the model's performance in real-time. This will be achieved by live streaming video from the Zed camera, however, under the same lighting conditions and background as the training data was recorded in. The model's ability to maintain high accuracy will be a determinant of its readiness for deployment and for testing in different conditions.

Criteria for Success in Real-Time Testing

The success benchmark for real-time testing is established by maintaining an accuracy level close to or above 90%. Should the model meet these criteria, it would indicate that the system is robust enough to be sent to Unity for integration. Figure 7 provides an overview of the results obtained from the real-time evaluation of the gesture recognition system.

Test	Accuracy
Real-Time Evaluation	93%

(a) Displays the result of the real-time evaluation, showing an overall accuracy of 93%. This confirms that the system meets the established criteria for integration.



(b) Practical application of the system in Python. The visualization shows the detected skeletal structure of the user and the identified actions in real time

Figure 7: Results and application of the gesture recognition system

Conversion to ONNX for Unity Integration

Upon successful validation in real-time testing, the model is converted into the Open Neural Network Exchange (ONNX) format. This step is necessary for integration with the Unity engine, which requires the model to be in ONNX format for compatibility reasons.

4.5 Unity Integration

Zed SDK provides a variety of different plugins for integrating complex systems with different development platforms. This integration process is fundamental for creating a seamless interactions between the application's logic and the camera's sensory data. Central to this integration is the `ZEDManager`, a class provided by the ZED Unity plugin that simplifies the camera's management and serves as a gateway for its features within Unity. It performs multiple functions ranging from camera initialization, retrieving images, mapping spatial data, and importantly, tracking skeletal movements, which in our case is essential for gesture recognition. Figure 8 shows the ZED Unity plugin correctly integrated and how the users skeleton joints have been translated into a 3D character.

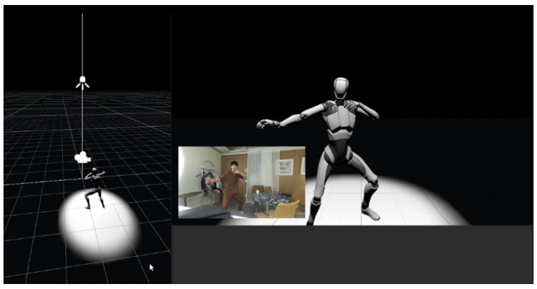


Figure 8: A visual representation of the system presented in Unity.

To utilize the `ZEDManager` and its foundational functions, the `zedcameraHandler` MonoBehaviour extends its processing on skeleton tracking, focusing on processing and interpreting gesture data. This is a core component in converting the camera input into actionable data within the Unity framework.

Initialization and Event Subscription

Upon the start of the application, the zedcameraHandler subscribes to two key events from the ZEDManager: OnZEDReady and OnBodyTracking. OnZEDReady ensures that the camera is properly initiated before the application begins processing data, providing a reliable starting point for tracking. furthermore, the OnBodyTracking event is where the majority of the systems action takes place as it receives data frames containing the skeleton information of tracked body keypoints.

Data Processing Pipeline

When a body is detected and tracked, the 38 keypoints are converted from ZED's 2D coordinate system to Unity's coordinate system using the ConvertZED2DToUnity method. This normalizes the data to Unity's environment considering its coordinate system and screen resolutions. The script stores a queue of the latest 30 frames worth of keypoints to form a sequence. This sequence is vital for understanding the temporal context of the movement, allowing the system to accurately recognize the specific action that is being presented.

Interaction with the Gesture Recognition Model

After collecting the desired number of frames, the data is preprocessed to correctly be fed into the model. It is converted into a Tensor which is the core data structure used by the Unity Machine Learning Agents Toolkit for neural network computations (Unity Sentis). The process takes place within the ZedManager_OnBodyTracking method, where the input is sent to the preloaded model, imported as an ONNX file, using Sentis API.

The models executes the prediction on a GPUCompute worker for performance, analyzes the sequential data and generates an output tensor that holds the probabilities of different actions detected within the sequence. The output is then analyzed to determine the most probable action performed by the user with a threshold of 50% before considering an action. In the case where none of the actions reach above the target, the system won't output a new action state.

Real-time Feedback and State Management

The given results are then translated into a user-friendly format, updating the UI elements in real time to present the counts of different actions such as kicks and boxing movements. This feedback loop is vital for assessing the accuracy of the interactive applications giving a instant response on the action. The zedcameraHandler works in collaboration with the state machine pattern, which will be detailed in the next segment. This joint system allows for a approach to accurately get responses on the predictions made and the probability distribution its bases its assessment on.

5 Results

This section presents an overview of the system and its integration within the Unity environment (Section 5.1), a detailed explanation of the Unity state machine used for testing and evaluating the system (Section 5.2). It then provides quantitative results from the system's performance based on the evaluations (Section 5.3) and concludes with an analysis of the model's performance metrics and confusion matrices (Section 5.4).

5.1 Unity State Machine

To test the system accuracy within the Unity environment, a state machine is integrated to handle different states of user interaction. A state machine, which is a design pattern used to manage states and behaviors, helps structure the pattern in which action recognition tasks such as kicking and boxing are recognized.

The state machine is made up of three classes, each implementing the `IState` interface, which prescribes `Enter`, `Execute`, and `Exit` methods. This interface allows for a consistent structure for state transitions, ensuring a clean entry and exit from one state to another, and that behavior is properly executed while the state is active.

At the heart of the Unity implementation is the `StateMachine` class, which contains references to the `ZedcameraHandler` instance and UI elements for displaying action counts. The class is responsible for managing the executions, state transitions, and updating the latest counts of detected kicks and boxing movements when leaving the current state.

Upon initialization in the `Start` method, the state machine transitions to the `StartState`, where it begins a monitoring phase for non-fighting activity. If 10 seconds pass without the `zedcameraHandler` detecting any fighting actions, the state machine moves to `notFighting State` where it can transition to a counting state, `CountingKicksState` or `CountingBoxing State`.

The counting states (`CountingKicksState` and `CountingBoxingState`) will increment the related action counter when exiting the state. These states also monitor changes in the detected action and transition to the appropriate state when a different action is observed.

The idle state `NoFightingState` is set for scenarios when no specific action is detected. This state serves as a fallback, keeping the system ready to transition to counting states upon the detection of relevant actions.

Figure 9 represents the state transition diagram, which illustrates the flow and relationship between the idle state and the kick state within the Unity state machine, the boxing state has a similar relation to kick state as they are both defined as counting states. The arrows in the diagram indicate the transitions between these states based on the detected actions and specified conditions.

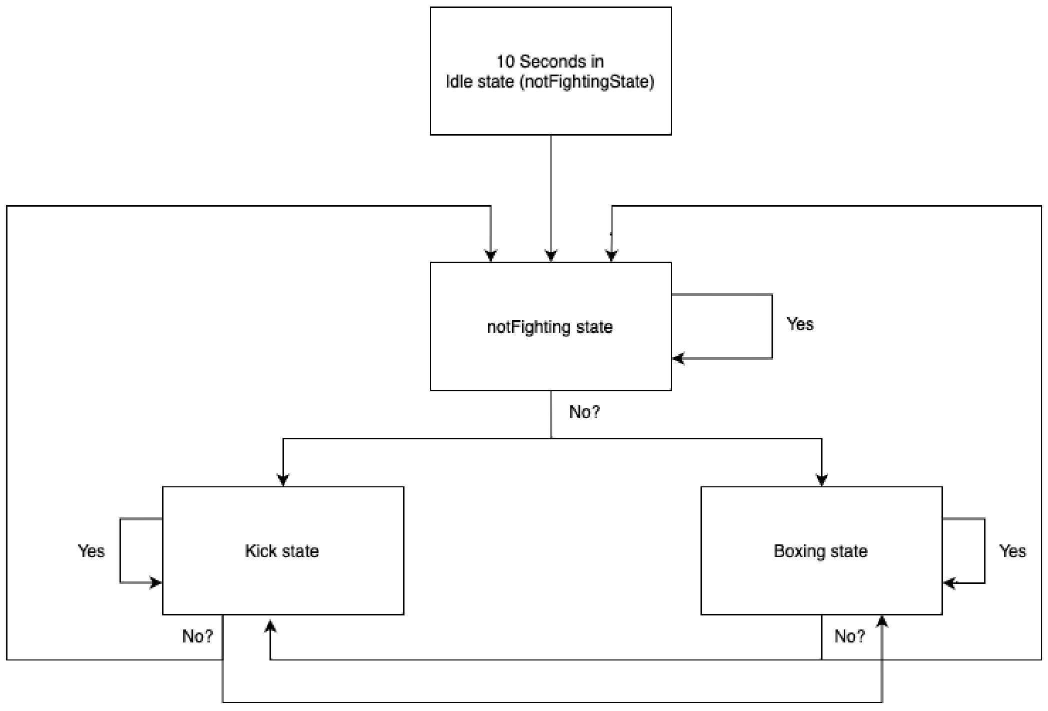


Figure 9: State transition diagram for the Unity gesture recognition state machine.

- StartState: This is the initial state where the user needs to stay in for 10 seconds before transitioning to `notFightingState`.
- notFightingState: The fallback state where no specific action is detected.
- CountingKicksState: state activated when a kick is detected. Transitions back to `notFightingState` if no action is detected for a specified duration or to `CountingBoxingState` if a boxing action is detected.
- CountingBoxingState: state activated when boxing is detected. Transitions back to `notFightingState` if no action is detected for a specified duration or to `CountingKicksState` if a kicking action is detected.

5.2 Quantitative Results

Table 3b & 3a provides a summary of the activities tested and the averaged overall performance of the gesture recognition model using two key metrics: accuracy and F1 score. The system was evaluated across four test cases that included various combinations of kicks and boxing activities:

- **Test 1:** Focused on 10 boxing actions.

- **Test 2:** Focused on 10 kick actions.
- **Test 3:** Combination of 10 kicks or boxing actions.
- **Test 4:** Combination of 10 kicks or boxing actions.

Table 3

Metric	Score
Accuracy	0.91
F1 Score	0.91

(a) Averaged summary of the model's performance metrics.

Test Number	Activity Counts
1	10 Boxes
2	10 Kicks
3	10 Boxes or Kicks
4	10 Boxes or Kicks

(b) Summary of the activities during tests.

5.3 Model Performance

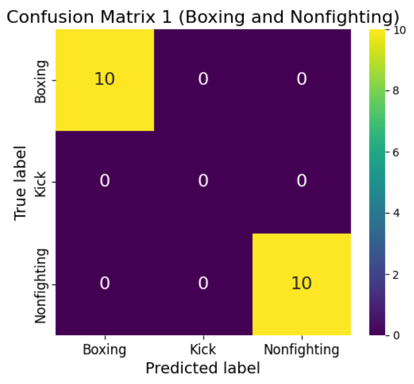
The performance of the gesture recognition model was evaluated based on the same metrics as before, the accuracy and F1 score. An average accuracy of 91% and an F1 score of 91% was recorded, indicating a high precision and recall. This suggests that the model is effective in recognizing defined gestures that it is trained on. To correctly asses the desired action, the system had to go back to its idle action "Nonfighting" between each gesture, resulting in every confusion matrices having just as many predicted "Nonfighting" values as the tested actions.

Figure 10 provides a visual representation of the model's performance for the four different tests. The matrices are interpreted as follows:

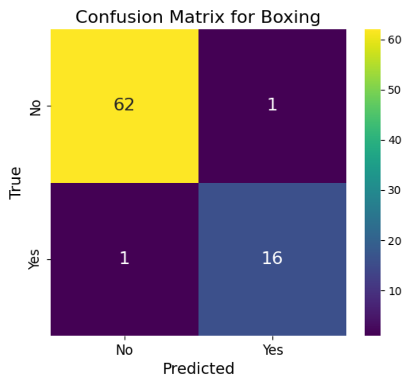
- **(a) Boxing and Nonfighting:** Represents the classification performance of the model on the test focused exclusively on boxing actions. The results indicate that the system correctly identified both boxing and nonfighting actions when they occurred, with 10 correct identifications for boxing and 10 for nonfighting. This is shown by the diagonal values in the confusion matrix, where the True label matches the Predicted label.
- **Confusion Matrix (b):** Represents the total classification of Boxing throughout all of the tests. Indicating that the model correctly classified True Negative 62 times and True Positive 16 times. There were 2 instance of False Positive and False Negative indicating a high accuracy.
- **Confusion Matrix (c):** Represents the classification performance on the test focused exclusively on kick actions. The results show four misclassifications of kick actions but correctly identify nonfighting actions 10 times. The primary misclassification occurs between kicks and nonfighting, with three instances of kicks being mistaken for nonfighting.
- **Confusion Matrix (d):** Represents the total classification of Kick throughout all of the tests. Indicating that the model correctly classified True Negative 56 times and

True Positive 18 times. There were 6 instance of False Positive and False Negative indicating a strong accuracy but with room for improvement.

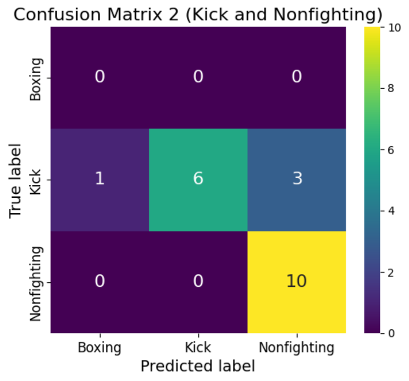
- **Confusion Matrix (e):** Represents the total classification of Kick and Boxing throughout test 3 and 4 tests. Indicating that the model correctly classified boxing 6 times, Kick 12 times and non fighting 20 times. There were 2 missclassifications on both counting actions.
- **Confusion Matrix (f):** Represents the total classification of nonfighting throughout all of the tests. Indicating that the model correctly classified True Negative 36 times and True Positive 40 times, giving a strong overall accuracy score.



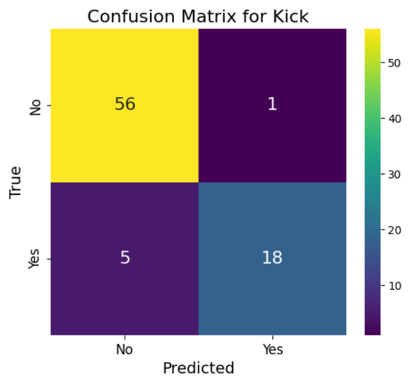
(a) Confusion Matrix 1 (Boxing and Nonfighting)



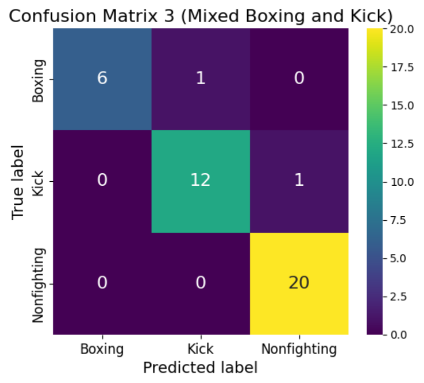
(b) Confusion Matrix 2 (Total Boxing)



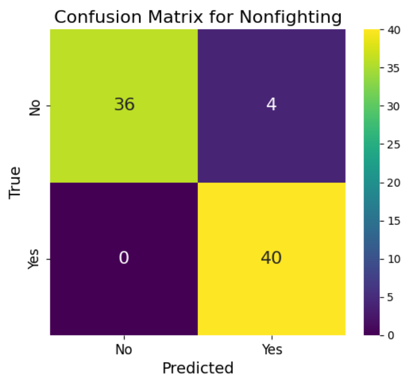
(c) Confusion Matrix 3 (Kick and Nonfighting)



(d) Confusion Matrix 4 (Total Kick)



(e) Confusion Matrix 5 (Mixed Boxing and Kick)



(f) Confusion Matrix 6 (Total nonFighting)

Figure 10: Confusion matrices representing the model performance.

6 Discussion

This chapter provides a detailed analysis of the model's performance and discusses the observations and limitations encountered during the study (Section 6.1), then about the importance of skeleton keypoints in action recognition (Section 6.2). Explains the model-based importance (Section 6.2.1) and data-based importance (Section 6.2.2), and finally, analyzes the importance of specific actions such as kicking, boxing, and non-fighting (Section 6.2.3).

6.1 Model performance

The model presents commendable results, comprising an average accuracy score of 91% and an F1 score of 91%. These metrics indicate an effective solution with efficiency in recognizing trained actions. The system present good results even when being transferred between multiple pipelines of the subsystem. However, there are some notable observations and limitations in the current implementation that merit discussion.

Initially, it was observed from the testing that the model would occasionally hallucinate between the actions non-boxing and kicking, in particular by a noisy environment in which the actions were performed. This suggests that the models can sometimes predict actions that are not present, most likely due to environmental cues that were not varied during the training phase. This sensitivity to the environment is a potential area for improvement in future iterations.

Another notable limitation in our experimental approach was the lack of variety in the training data. All the data used to train the model were collected in the same location and on the same day. This homogeneity in how the dataset looks sets up the model for disruptive environmental factors later when variations in lighting and background noise gets presented. The models accuracy in varied real-world scenarios remains untested.

The primary objective of this project was to establish a functional pipeline for gesture recognition rather then finetuning and perfecting a model across multiple actions. This focus was chosen to prioritize the development of a robust machine learning system that integrates with the Unity environment.

6.2 Explaining Skeleton Keypoint Importance in Action Recognition

To get a better understanding of the decision-making process in our system, we thoroughly analyze the significance of each skeletal keypoint. This provides a clearer explanation of how the system has adjusted its trained parameters to inform its decisions.

6.2.1 Model-Based Importance Explained

Model-Based Importance is computed directly from the static weights assigned to each keypoint in the LSTM layer of the model. These weight reflect the importance of certain keypoints, giving a foundation to accurately understand the systems predictions across various actions. Since the importance is given by analyzing the model internal weights after training, it remains constant unless the model is retrained. Finding out which features are weighted more heavily by the model can guide developers in analyzing and understanding the system.

Visualization of Model-Based Importance

Figure 11 illustrates the normalized importance of each skeleton keypoint as determined by the static weights in our LSTM model. This visualization aids in quickly identifying which features are deemed most critical by the model for the recognition of diverse actions within the Unity environment.

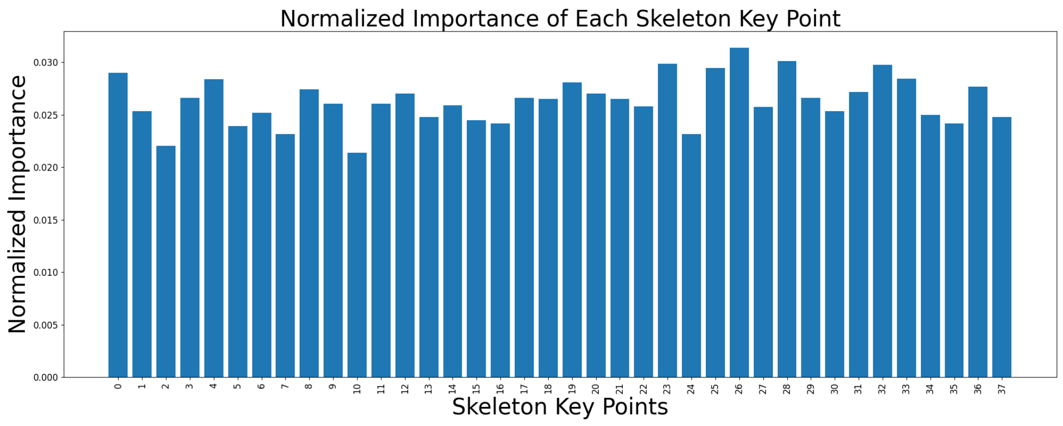


Figure 11: Normalized importance of each skeleton keypoint based on the LSTM model static weights.

Figure 11 presents a well-balanced distribution across the 38 keypoints used for body tracking[1], a notable observation is that the key points associated with hands (keypoints 32,33), ankles (keypoint 23) and feet (keypoints 25,26) seem to have a higher importance (>0.027). This is likely due to the nature of the actions like kicking, boxing, or non-fighting that the model is trained upon. The hands (left and right) have a high involvement in upper body boxing movements, ankles and feet are crucial for dynamic lower body movements such as kicking. These patterns are further seen in other keypoints such as the clavicles (keypoint 10) and spine (keypoint 2), which exhibits lower importance scores (<0.023) as they are not as crucial for the trained actions. Although the values are not particularly large, this gives us a fundamental basis for refining the feature selection and model training strategies for future iterations of the model.

[1] `https://www.stereolabs.com/docs/body-tracking`

6.2.2 Data-Based Importance

Furthermore, the Data-Based Dynamic Importance is calculated by ablating features in the ‘X_test‘ dataset and measuring the resultant decrease in model accuracy. In the context of machine learning, ablating refers to systematically removing or deactivating certain features of the model to study and understand the impact of each feature on the performance. This is done by comparing the accuracy of the model on the dataset and calculate the difference when a specific feature is removed, this is then done for every feature available. The Data-Based Importance is calculated by ablating features in the ‘X_test‘ dataset but provides a dynamic measure of feature significance that can adapt based on the dataset used for calculation.

This dynamic importance assessment plays a fundamental role in further fine-tuning the model sensitivity to key features, specially in cases where more actions are added to the task.

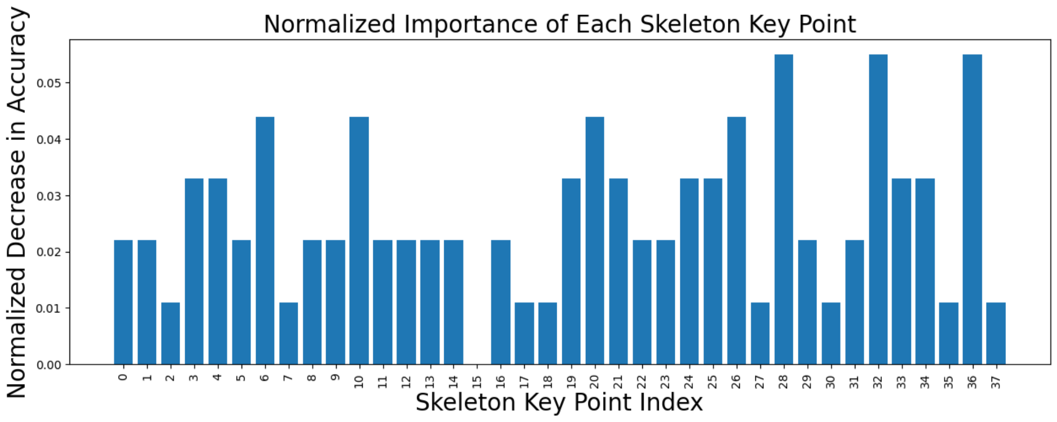


Figure 12: Normalized decrease in accuracy for each skeleton key point, illustrating the impact of keypoint ablation on model performance.

Figure 12 illustrates the normalized decrease in accuracy across various key points when individually ablated. Notably, no significant changes are observed in any particular key point, as the greatest impact on the model accuracy is only about a 0.05% difference. This underscores the model stable predictive capabilities. However, in context, features related to the left side of the body, including the left heel (key point 28), left hand (key point 32), and left pinky (key point 36), appear to play a critical role in recognizing the actions. This could be due to the fact that most kicks recorded in the training data were done with the right foot, and most jabs were executed from an orthodox stance[2], making the left side crucial for distinguishing between non-fighting, kicking, and boxing actions.

6.2.3 Importance Across Specific Actions

For action-specific investigations such as **kicking and boxing**, the dynamic importance calculation revealed certain key points as particularly significant. In the kick action, the results given in Figure 13 are consistent with expectations, as all the crucial key points are related to the lower body: right hip (key point 19), right big toe (Key point 25), left small

[2]https://en.wikipedia.org/wiki/Orthodox_stance

toe (key point 26), and left heel (key point 28).

However, for the boxing stance, the results shown in Figure 14 indicate that the left big toe (key point 24) has a critical role in accurately predicting the action. This sensitivity is unexpected, as arm movements are more pronounced and should be considered key features. This discrepancy suggests that the model may require fine-tuning to better recognize upper body movements characteristic of the boxing stance. Fine-tuning could involve adjusting the training data to emphasize arm and shoulder movements or modifying the model's architecture to capture more nuanced information related to upper body dynamics.

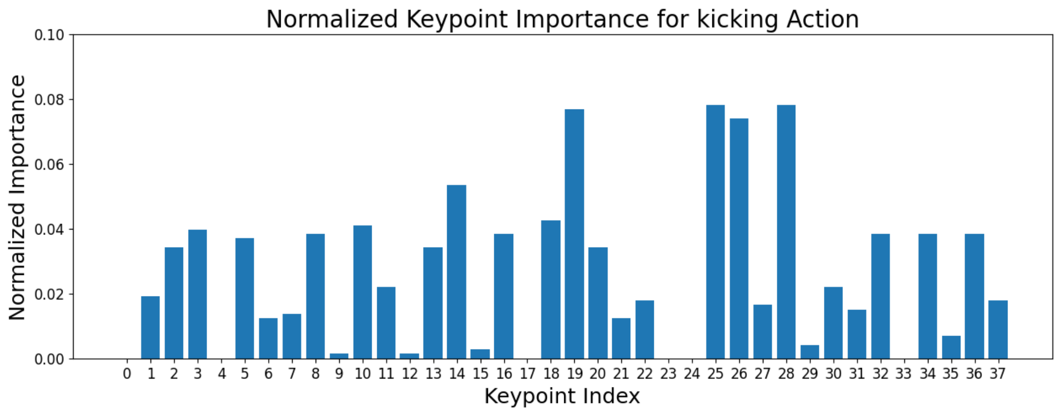


Figure 13: Normalized Keypoint Importance for Kick Action

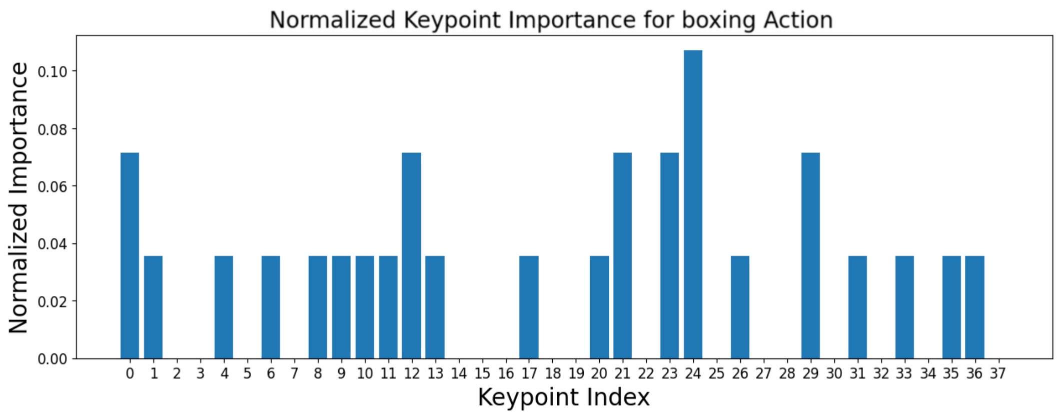


Figure 14: Normalized Keypoint Importance for Boxing Action

Non-fighting actions, which involve more subtle and varied movements, showed a distributed importance across several left body keypoints in Figure 15: left heel (key point 28), left hand (keypoint 32), and left hand pinky (keypoint 36). This distribution suggests the characteristics of the left body movements are crucial for categorizing the training data. It also explains the frequent confusion (hallucination) the system exhibited between kicking and non-fighting actions, as both share key features within similar regions of the body.

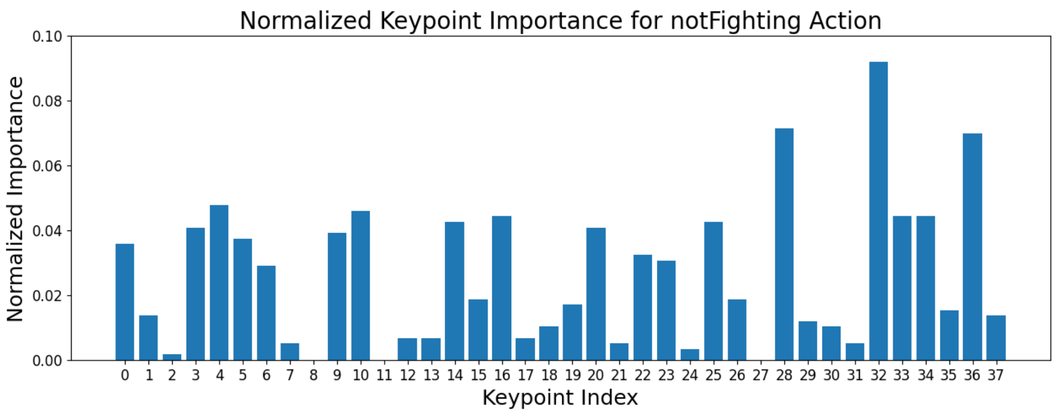


Figure 15: Normalized Keypoint Importance for Non-Fighting Action

7 Conclusion and Future Work

This chapter summarizes the findings of the research (Section 7.1), discusses the contributions made to the fields of human-computer interaction (HCI) and interactive gaming (Section 7.2). It also addresses the limitations encountered during the study (Section 7.3) and lastly, provide suggestions for future research to enhance the system further (Section 7.4).

7.1 Summary of Findings

The goal of this master thesis was to explore the integration of advanced human-computer interaction (HCI) techniques within a virtual environment, giving an architecture that could be further finetuned for games or 3D installations. This would provide a more intuitive and immersive interaction without the need for traditional physical controllers. Using skeleton tracking and machine learning algorithms, we developed a successful pipeline capable of real-time gesture recognition, providing an innovative solution and step forward for interactive gaming, as for now, no other clear solution has been found on integrating action recognition in Unity. The findings open up new possibilities for game design, by providing an new interaction paradigm within Unity Engine, game developers can infer players movements and effectively utilize these recognized actions.

7.2 Contributions to HCI and Interactive Gaming

The advancements made through this master thesis contribute to the fields of HCI and gaming. Prior to this research, there was no known action recognition pipeline integrated with the Unity Engine, as far as we are concerned. By utilizing new technology and machine learning, particular a trained Long Short-Term Memory (LSTM), this system has not only functioned effectively but also presented scores higher than expected in accuracy. It is important to note that these scores were achieved through testing on a single user, which may have influenced the results. A diverse user base gives the system variability in length and movement patterns, which might challenge its efficiency, this is crucial to take into consideration when reflecting the performance as the training data was generated by the same user.

7.3 Limitations

This study also acknowledges certain limitations. The system's performance is sensitive to environmental variations, such as changes in lighting and background. Built on the Zed2i camera's skeleton tracking and Zed SDK, the architecture would require modifications to integrate with other systems. As for now, it would also require some code adjustments to add actions or remove them, this would require a user to have some coding knowledge. While the system is fast, its response time is not close to meeting the real-time requirements of fast-paced applications like first-person shooter games, where minimal latency (< 50ms) is crucial[1].

7.4 Suggestions for Future Research

Future work on this project could explore several enhancements, testing the model under different lighting conditions and with background noises could help in understanding and improving its robustness and accuracy in more diverse environments, fine-tuning and increasing the training dataset or potentially use a pretrained model would yield to consistent results across multiple actions. This would not only increase the models generalization capabilities, but also help with environmental noise. With having the Zed SDK incorporated, there are also possibilities to cross trained the model on depth map images or spatial mapping, giving multiple possibilities in creating a resilient and versatile gesture recognition system. Currently, the system uses 2D data from skeleton tracking. A significant enhancement would involve adapting the system to extract and process 3D data points, which could improve both accuracy and processing speed.

In addition, future user testing can be carried out to further evaluate the system and the trained model. This would give insight into the performance of the model in real-world scenarios, allowing for valuable information that can help in adjusting the model to better aligned with the needs and expectations of the system. This iterative process of user testing and model improvement is fundamental in a architecture that is developed to be robust and user-friendly in the future.

To further develop the research done on the on the system keypoint importance, it is vital to conduct future testing with multiple users. As for now, the focus has been on building a functional pipeline that operates from training to predicting movements in Unity. Optimizing the model's ability to recognize these actions has been a secondary priority, given that the current model is trained on only 30 sequences of specific actions, each 1 second long.

For a more advanced system, it would be valuable to compare the importance of keypoints and its correlation across different users. Key questions to investigate include how the system adapts to each new user and whether the keypoint importance aligns with the findings in this research, indicating a generalized pattern. Researching these areas would provide a deeper insights into the model's decision-making process. Exploring the importance of keypoints would increase explainability of the model, helping demystify the model's internal workings and create transparency in any biases that might exist.

Finally, we aim to publish this work as a Unity add-on, making it available and open source for anyone to use. This will not only validate the system's effectiveness but also further improve research on gesture recognition within the Unity community.

[1] `https://www.nvidia.com/en-us/geforce/news/reflex-low-latency-platform/`

References

Bowling, M., Fürnkranz, J., Graepel, T., & Musick, R. (2006). Machine learning and games. *Machine Learning*, *63*, 211-215.

Cheng, Q., Cheng, J., & et al., Z. R. (2023). Multi-scale spatial–temporal convolutional neural network for skeleton-based action recognition. *Pattern Analysis and Applications*, *26*, 1303-1315. doi: 10.1007/s10044-023-01156-w

Dhakecha, H. (2022, 08). A methodological study of human-computer interaction: A review. *International Journal for Research in Applied Science and Engineering Technology*, *10*, 195-200. doi: 10.22214/ijraset.2022.46127

Fatima, S., Sangeetha, G., Ponmurugan, P., Arularasan, A. N., Prabhu Chakkaravarthy, A., Denis, R., & Chinnasamy, A. (2023). Machine learning development in solving critical medical problems. In *2023 International Conference on Inventive Computation Technologies (ICICT)* (p. 186-191). doi: 10.1109/ICICT57646.2023.10134418

Gaikwad, S., Ghodekar, R., Gatkal, N., & Prayag, A. (2023, 05). Human action recognition using deep learning. *International Journal for Research in Applied Science and Engineering Technology*, *11*, 1888-1892. doi: 10.22214/ijraset.2023.51960

Gill, K. S., Sharma, A., Anand, V., Sharma, K., & Gupta, R. (2023). Human action detection using efficientnetb3 model. In *2023 7th international conference on computing methodologies and communication (iccmc)* (p. 745-750). doi: 10.1109/ICCMC56507.2023.10083926

Goudelis, G., Karpouzis, K., & Kollias, S. (2013). Exploring trace transform for robust human action recognition. *Pattern Recognition*, *46*(12), 3238-3248. doi: https://doi.org/10.1016/j.patcog.2013.06.006

Homolak, J. (2023, 02). Opportunities and risks of ChatGPT in medicine, science, and academic publishing: a modern Promethean dilemma. , *64*, 1-3. doi: 10.3325/cmj.2023.64.1

Moussa, M. M., Hamayed, E., Fayek, M. B., & El Nemr, H. A. (2015). An enhanced method for human action recognition. *Journal of Advanced Research*, *6*(2), 163-169. doi: https://doi.org/10.1016/j.jare.2013.11.007

Nguyen, C., Nguyen, N., Huynh, S., Nguyen, V., & Nguyen, S. (2022). Learning Generalized Feature for Temporal Action Detection: Application for Natural Driving Action Recognition Challenge. In *2022 ieee/cvf conference on computer vision and pattern recognition workshops (cvprw)* (p. 3248-3255). doi: 10.1109/CVPRW56347.2022.00367

Rahul Rai, D. I., Manoj Kumar Tiwari, & Dolgui, A. (2021). Machine learning in manufacturing and industry 4.0 applications. *International Journal of Production Research*, *59*(16), 4773–4778. doi: 10.1080/00207543.2021.1956675

Rashid, M., Jain, S., & Yadav, R. (2023, 05). Review paper on professional issues in human computer interaction. *International Journal of Advanced Research in Science, Communication and Technology*, 50-57. doi: 10.48175/IJARSCT-10720

Vanshika, & Gupta, N. (2022). Machine learning applications in healthcare. In *2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)* (p. 1-6). doi: 10.1109/ICRITO56286.2022.9964865

Victorelli, E. Z., Dos Reis, J. C., Hornung, H., & Prado, A. B. (2020). Understanding human-data interaction: Literature review and recommendations for design. *International Journal of Human-Computer Studies*, *134*, 13-32. doi: https://doi.org/10.1016/j.ijhcs.2019.09.004

Yuna, H., Nakamura, E., Kageyama, Y., Ishizawa, C., Kato, N., Igarashi, K., & Suzuki, M. (2022). Development of action-recognition technology using lstm based on skeleton data. *Journal of Advanced Research in Human-Computer Interaction Studies, Akita University*, 101-110.

A First Appendix

For further reference and access to the project's code and additional resources, please visit the following links:

- GitHub Repository for Unity
- GitHub Repository for ActionDetection