Multi-modal Feature Fusion Using Full Sequences for Dynamic Hand Gesture Recognition with Simulated Robotic Arm Control

Parsa Sheykholeslami

Thesis submitted to the University of Ottawa in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

School of Electrical Engineering and Computer Science

Faculty of Engineering

University of Ottawa

March 2026

© Parsa Sheykholeslami, Ottawa, Canada, 2026

Abstract

Dynamic hand gesture recognition (DHGR) enables accessible human-robot interaction by interpreting sequential human hand movements rather than static poses. Previous DHGR systems only focused on using the RGB modality in datasets and ignored depth. This thesis addresses this issue using a multi-modal classifier preserving temporal integrity. The InceptionV3-LSTM architecture is recreated, using a public RGB-depth dataset of six dynamic gestures. Full 40-frame sequences are used along with stratified 5-fold cross-validation to prevent sequences splitting across folds. The feature extraction pipeline fuses visual and landmark features from both RGB and depth modalities in parallel InceptionV3 streams, feeding a stacked LSTM-RNN. The results demonstrate that overfitting is reduced when using full-sequence multi-modal training, with validation loss decreasing while exceeding RGB-only accuracy. This work contributes a multi-modal pipeline for DHGR that is implemented in a simulated robotic control application.

Acknowledgments

I will be forever grateful to my supervisors, Dr. Emil M. Petriu, and Dr. Hilmi Dajani, for giving me the opportunity to pursue a higher education. Without their guidance, support, and insight, this thesis would never have been possible. Since my undergraduate years I have developed huge respect for them by being a part of their classes, and it has been an honor to have them back me and my work.

I would like to thank a fellow student and Ph.D. candidate, Sajad Sadeghkhani, for his help in teaching me the basics of The Digital Research Alliance of Canada and MobaXterm, without which this thesis would have taken twice as long when I first started.

Finally, I would like to thank the best parents in the world, my mom and dad. Throughout my life it always felt as if they had worked their whole lives to give me the opportunity to pursue higher education. Even when I did not want to do it, they pushed me with nothing but love and devotion and this achievement is for them.

Table of Contents

List of Figures

List of Tables

List of Acronyms

CNN – Convolutional Neural Network

CSV – Comma-Separated Values

DHGR – Dynamic Hand Gesture Recognition

HAN – Hierarchical Attention Network

HCI – Human Computer Interaction

HGR – Hand Gesture Recognition

IWBC – Improved Weber Binary Coding

LFD – Low Frequency Descriptor

LMC – Leap Motion Controller

LSTM – Long Short Term Memory

RGB – Red Green Blue

RNN – Recurring Neural Network

ROC-AUC – Receiver Operating Characteristic Area Under the Curve

SPM – Semantic Pose to Motion

SVM – Support Vector Machine

URDF – Unified Robot Description Format

1. Introduction

Hand gesture recognition (HGR) has gained significant attention in the past decade as a significant application in several fields such as robotics, virtual reality, and human-computer interaction (HCI). HGR enables machines to understand and respond to gestures performed by humans, facilitating seamless interactions while also providing more accessible and user-friendly technology.

HGR is traditionally used for applications requiring simple static hand gestures such as sign language, a “thumbs up” or “okay”. Through advancements in machine learning and computer vision, machines capable of recognizing dynamic hand gestures have been developed (Tripathi and Verma, 2023). Dynamic movements such as moving a hand from left to right are significantly more challenging for a machine to recognize due to their sequential and temporal nature. The machine is required to remember the sequential order of the continuous movement to achieve accurate recognition (Kopoklu et al., 2020). Despite recent advancements, there are still barriers to overcome, particularly in achieving robust and accurate recognition in real-world environments where variations in hand shape, orientation, and movement patterns are common.

In the growing fields of robotics, machine learning, and computer vision, dynamic hand gesture recognition has several possible applications. Specifically, there are many new applications in fields such as augmented or virtual reality, gaming, and accessibility for disabilities. In the medical field, dynamic hand gestures may be utilized to enhance surgeries and training. In the automotive field, driver safety can be improved by allowing hands-free control of some in-car systems (Csonka et al., 2023).

In dynamic hand gesture recognition (DHGR) applications, machine learning models are required to be robust and accurate to handle variations in hand shape, orientation, and movement patterns. To achieve this, the approach of relying on an individual frame is insufficient. Instead, recognizing the entire instance of a gesture comprising several sequential frames is required. The dataset, provided by Jeeru et al. (2022) contains a combination of 29,718 frames consisting of RGB and depth modalities capturing various instances of six different hand gestures performed by multiple individuals in different environments. A Long Short-Term Memory (LSTM) - Recurring Neural Network (RNN) classifier can leverage the sequential nature of the dataset, especially when built

for the InceptionV3 pre-built Convolutional Neural Network (CNN) model (Hax et al., 2024). Several approaches have been used with DHGR, such as 2D CNNs, 3D CNNs, hybrid CNN+RNNs, two-stream models, and more recently transformers, all of which are described in more detail in the literature review. This study is careful in the approach taken to model training, validation, and testing. Through sequence-based validation, this research ensures temporally adjacent frames from the same gesture in any modality do not appear in both training and test sets. The model learns the temporal patterns of each gesture sequence using both the RGB and depth data available in the dataset.

A practical significance of this study is in the DHGR application of simulated robotic control. For instance, Gourob et al. (2021) demonstrates real-time static hand gesture recognition to control a robotic hand's fingers. Adapting this approach to dynamic hand gestures can unlock a broader range of motions and more natural human-robot interaction (HRI), especially when integrated with a robotic arm which includes an elbow-like joint. This advancement has the potential to enhance the usability of robots in various fields, such as assistive technology, teleoperation, and virtual reality.

1.1 Thesis Contributions

This thesis introduces a novel multi-modal camera-based real-time dynamic hand gesture recognition system for training a human-like robotic arm and hand to perform six different dynamic hand gestures thereby enhancing HRI. A pre-recorded dataset of dynamic hand gestures (Jeeru et al., 2022) was used to evaluate the accuracy of the proposed DHGR system compared to previous HGR approaches.

This thesis builds directly on the hybrid LSTM-RNN architecture introduced by Hax et al. (2024) and other similar studies such as Yaseen et al. (2024). The main contributions are as follows:

- Demonstration of improvement in performance by using both available modalities in dataset through multi-modal fusion. In previous studies, Hax et al. (2024) achieved 83.7% test accuracy and Yaseen et al. (2024) achieved 89.7% by training the model on every fourth frame in each sequence, i.e., ten out of forty frames per sequence and only using the RGB modality from the RGB-Depth dataset. These single modality methods could not utilize the full strength of the multi-modal dataset. Expanding on their work, training is

executed using the full 40-frame sequences with sequence-based stratified 5-fold cross-validation while both RGB and depth modalities are used in a multi-modal feature fusion.

- Ensuring no multi-modal leakage by matching RGB and depth versions of each sequence to avoid splitting the modalities of the same sequence across training and testing.
- Proof of concept for simulated robot implementation. The dataset's gestures are mapped to joint trajectories for a robotic arm in a simulated environment, allowing the robot's gestures to mimic the gestures performed by humans in the dataset.

1.2 Thesis Outline

This thesis asks how does multi-modal full-sequence training impact generalization. It develops a dynamic hand gesture recognition system utilizing a pre-trained InceptionV3 CNN model combined with an LSTM-RNN classifier. The work achieved accurate state-of-the-art classification of the six different hand gestures in the dual-modality depth and RGB dataset, and introduced a novel approach to multi-modal feature fusion, enhancing the model's ability to use both RGB and depth data.

This thesis has five chapters (other than this introduction) that cover the following:

- **Chapter 2** presents the background and literature review that highlight many aspects of hand gesture recognition systems. The method of gathering data and inputs, approaches for static versus dynamic gestures, different modalities, various deep learning architectures, and robotic control are covered.
- **Chapter 3** presents the detailed end-to-end pipeline used in experimentation. The dataset and its processing, software environment, feature extraction, learning algorithm, stratified k-fold sequence-based split, and the control of the simulated robot are described along with their roles in the methodology.
- **Chapter 4** presents quantitative comparisons between the single stream RGB and multi-modal fusion four-branch approaches. Several plots are included such as training/validation, loss/accuracy, performance analysis, and confusion matrices. Additionally, this chapter shows trajectory plots and images of the simulated robot arm.
- **Chapter 5** discusses the impact of multi-modal fusion on the previously single modality model. Common misclassification patterns are analyzed, and the multi-modal method is

contrasted to the original RGB-only method. The simulated robotic control aspect of the study is discussed along with the study's limitations.

- **Chapter 6** presents the conclusions and suggests directions for future work.

2. Background and Literature Review

This chapter provides background information and a literature review of hand gesture recognition and the deep learning methods being applied in the field. Section 2.1 observes the traditional sensor-based and vision-based hand gesture recognition approaches, with them being covered in subsection 2.1.1 and 2.1.2 respectively. Section 2.2 differentiates between the concepts of static and dynamic hand gesture recognition with each being covered in subsections 2.2.1 and 2.2.2 respectively. Section 2.3 covers the different modalities that are often observed in hand gesture recognition such as RGB/depth and landmark coordinates in subsections 2.3.1 and 2.3.2, along with the fusion of these modalities in subsection 2.3.3.

2.1 Traditional Hand Gesture Recognition

2.1.1 Sensor-based Approaches

In HGR, sensor-based approaches refer to wearable or attached hardware such as gloves to measure finger positions or haptic feedback from a user rather than images. The early sensor-based systems achieved high accuracy in detecting gestures; however, they were costly, and a lot of calibration was required for the glove-based devices. In Zhu (2020), the *CyberTouch* data glove seen in Figure 2.1 recorded a human operator's real-time bone-joint angles in the hand and provided linear vibration feedback to the fingers and palm. The glove featured eighteen embedded sensors which are flexible. Each of these sensors were paired and positioned in between the joints of each finger. Additionally, one sensor was placed between the intermediate phalange and the proximal phalange, while another was placed between the latter and the metacarpal joint. Along with the help of 6 built-in actuators, the eighteen joint sensors provided haptic feedback to the human operator's hand. These 6 actuators vibrated as a form of haptic feedback and were placed in each fingertip and in the palm. The hands were detected using a Leap Motion Controller (LMC). The *Ring Ada* robotic hand in Zhu (2020) was developed as an upgrade to the open-source Ada hand platform and is what the hand gesture tracking was applied to, so that the robotic hand would be able to perform grasping while providing haptic feedback to the operator. In Zhi (2018), Microsoft Kinect-based systems were used since they are capable of sensing depth and hand gestures although they are more suitable for full-body tracking compared to LMC's hand tracking specialization.

In Lin et al. (2024), real-time monitoring of surface electromyography (sEMG) signals prompted researchers to develop sEMG-based gesture recognition algorithms, proposing a gesture recognition model that leverages multi-channel sEMG signals captured by an sEMG sensor. The model was evaluated and validated using a dynamic gesture dataset captured by a wearable sensor. The model achieved a relatively balanced compromise between prediction accuracy and processing speed compared to previous works.

Purely sensor-based approaches that allowed for more accurate real-time hand gesture tracking were limited due to the sensor logistics such as the need for wiring and in some cases a glove that needed to be worn. In the following section vision-based approaches are described that do not require any worn sensors for hand gesture tracking.

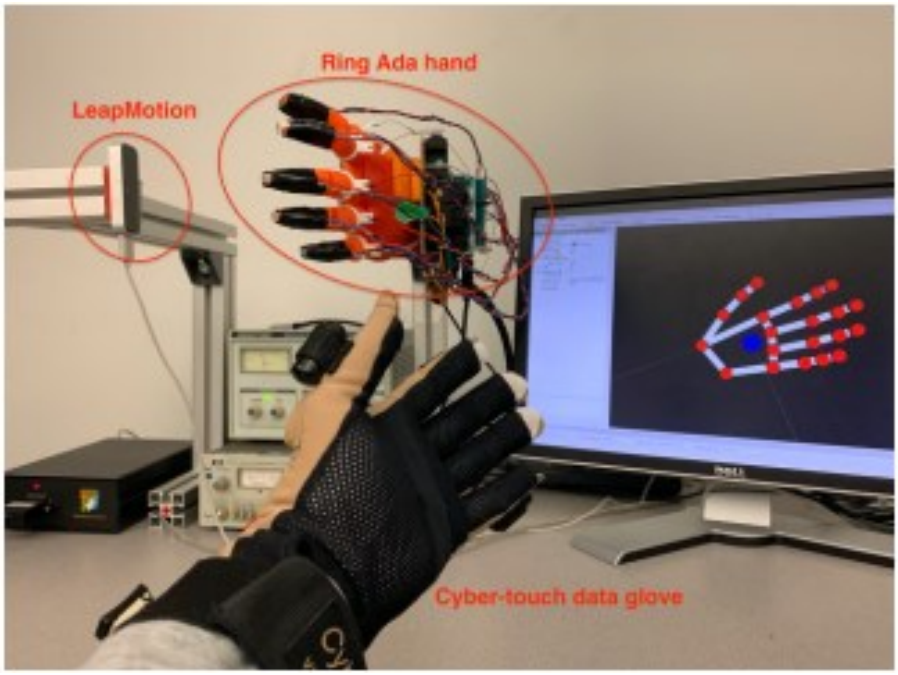

Figure 2.1 CyberTouch data glove setup (Zhu, 2020).

2.1.2 Vision-based Approaches

In Zhi (2018), an LMC was used. This technique captured precise 3D spatial data and tracked hand and finger movements with a high accuracy while having a limited working area. The LMC was considered the optimal depth sensor choice due to its portability, affordability, detailed tracking information, and connectivity. However, there remain limitations in this solution. The most

important is the limited working area of the LMC of 25 to 600 millimeters above the surface with a 150-degree field of view. Along with this, due to potentially weak USB contact or short breaks in the Leap Service process, communication interruptions or delays caused by occasional loss of signal may result in invalid tracking frames which affect gesture recognition accuracy.

One solution to these challenges may be to use a depth camera to capture hand gestures. In Sebbah and Chelali (2024), a Low Frequency Descriptor (LFD) and Improved Weber Binary Coding (IWBC), both vision-based approaches, were used to recognize static and dynamic hand gestures. The datasets were evaluated by SVMs and k-nearest neighbor classifiers with both descriptors using SVM and yielding a 95% recognition rate for American sign language. The authors propose that Deep Learning classifiers may be used to improve the accuracy of their hand gesture recognition system. In Sahoo et al. (2022), they explore RGB and RGB-D cameras that are used as vision sensors to observe hand gestures. The Intel RealSense D435 and D435i cameras were used to capture the dataset in Jeeru et al. (2022), and Xie et al. (2025) respectively. The transition to vision-based approaches in hand gesture recognition systems expanded its possible applications in human-computer interactions, augmented and virtual reality, and robotics. Below is a more detailed list of applications for vision-based hand gesture recognition.

- Clinical operations – Surgeons who are unable to use their hands are able to manage data from Magnetic Resonance Imaging (MRIs) and X-ray scans by just using gestures instead of having to touch anything (Oudah et al., 2020).
- Sign Language – People with hearing loss or who are mute are able to communicate with each other and other people through sign language which consists of many hand gestures. Human-Computer Interaction will be more accessible for these people (Gourob et al., 2021).
- Robotics – Robots can be controlled through hand gestures either with vision-based or sensor-based methods (Csonka et al., 2023).
- Home Automation – Hand gestures can be used to control smart home appliances such as lights, fans, curtains, thermostats, etc.
- Vehicle Automation – Hand gestures can be used to control the different features in modern vehicles, such as operating the music system, navigation, or phone calls (John et al., 2016).

2.2 Static and Dynamic Gesture Recognition

In the following section, vision-based approaches used in previous works are presented, and the focus is on how vision-based approaches evolved from static gestures to dynamic gestures.

2.2.1 Static Hand Gestures

Before the development of dynamic hand gesture recognition, the traditional method for classification of hand gestures was done using static gestures where the user holds their hand in a specific pose such as open palm or closed fist. When there is no significant motion of the hand over time, the recognition of the gesture boils down to a single-frame image classification problem (Csonka et al., 2023). Human hands have a well-defined anatomical structure of five fingers extending out from the palm, so different hand poses can be distinguished by the number of outstretched fingers, and their distances and placements in relation to each other, Thus, the approach to static hand gesture recognition focused on extracting geometric features from the silhouette of a hand (Li and Zhang, 2022). By computing the distance and orientations of the lines connecting the convex and concave extrema of the hand contours, a geometric descriptor along with a linear SVM were used for static hand gesture recognition (López-Casado et al., 2019). Similarly, concave points between fingers were detected using convex hull with the number of concave points and fingertips used to detect hand gestures.

Another approach was to extract a set of four hand gesture features, which included the distances of the fingertips from the hand center and the palm, the curvature of the hand contour and the geometry of the palm region (Dominio et al., 2014). These same features were combined into a single feature vector for hand gesture recognition (Marin et al., 2015). Such purely geometric features are robust in terms of lighting changes and interpretability; however, they struggle when faced with overlapping fingers or complex backgrounds. By using geometry-based normalizations and Krawtchouk moment features, the hand and forearms can be extracted through skin color detection and anthropometric measures, with rotation normalization for aligning the extracted hand, and Krawtchouk moment features to represent the hand gestures (Priyal and Bora, 2013).

Static hand gesture recognition is able to decipher the meaning represented by each class of hand gesture by processing the combining state of different fingers stretching out in an image. Traditionally, each static hand gesture is a combination of different fingers sticking out, with the

difference between each gesture being the difference in the number of fingers sticking out (Li and Zhang, 2022). The state-of-the-art neural network deep learning methods have proven to perform well on static hand gesture recognition by learning hierarchical features directly from the pixels of the images. This is seen in Satybaldina and Kalymova (2021) where they achieved an average accuracy of 99.4% for 6 static gestures, and in Deng et al. (2024) where they achieved an experimental accuracy of 96.62%. Binary hand gesture images were segmented and used to train a Convolutional Neural Network (CNN) and a stacked denoising autoencoder with both performing well on the same hand gesture dataset (Oyedotun and Khashman, 2016).

2.2.2 Dynamic Hand Gestures

Dynamic hand gestures extend from static gestures by introducing the temporal domain, considering sequences of movements rather than a static, held hand pose. Single-frame classifiers do not handle dynamic data well because a single silhouette or mask cannot capture a hand that is closing, opening, or moving side to side. For static gesture recognition, only one image frame is considered (Li and Zhang, 2022). In contrast, dynamic gesture recognition requires the system to have the ability to remember frames to interpret the trajectory of a gesture that spans multiple frames (Tripathi and Verma, 2023). The system must be able to recognize the order in which the frames are so that it can recognize the gesture. Single-frame models fail to capture the temporal features thus risking misclassification on gestures that look similar in isolated frames but differ in their motion.

Typically, when working with dynamic hand gestures, a video input is required to represent the movements and motions of the hand gestures. This video input is a sequence of image frames playing in rapid succession. For dynamic gestures, the spatiotemporal features must be extracted. There are several deep learning architectures that are designed for temporal data classification. The less complex model is the 2D CNN which has low computational cost and is commonly used for spatial data feature extraction and classification (Yaseen et al., 2024). The more complex model is the 3D CNN where the third dimension represents the temporal data. 3D CNNs can learn the spatiotemporal features from a multitude of feature maps, while 2D CNNs can only learn from one feature map (Hax et al., 2024). Then, there are two-stream CNNs which use the 2D CNNs for sparse data classification and a recurrent neural network (RNN) for the classification of the temporal data. These hybrid models are most commonly used due to their increased accuracy and

reduced computational cost, combining CNNs with RNNs, gated RNNs, or LSTM. These different models and approaches are discussed further in Section 2.4 on Deep Learning Architecture for Video.

While static gesture recognition has been thoroughly explored in real-time applications, proving to work successfully, there are still not enough accurate dynamic gesture recognition systems developed and proven to work in real-time applications (Zhi, 2018). For static gestures, real-time recognition is seen applied to robotics in Csonka et al. (2023) where a camera is attached to a mobile robot and used to detect static gestures performed by a user such as "forward", "backwards", and "stop". Once a static gesture is recognized by the robot's CNN, it moves according to the gesture. For real-time applications involving dynamic gestures, the robot would need a way to distinguish when a gesture begins and when it ends, paying attention to the temporal nature of the dynamic gestures compared to static ones.

2.3 Modalities

2.3.1 RGB and Depth

The red, green, blue (RGB) modality captures color information in images by separating incoming light into its respective red, green, and blue components. RGB cameras are specifically designed to capture the human visible light spectrum by using a color array filter with the standard Bayer BGGR pattern, which is sensitive to red, green, and blue, the primary colors of visible light for humans (TechNexion, 2025). The Bayer filter array is arranged in a mosaic on top of the camera's sensor pixels, dedicating each pixel to capture a specific color.

The depth modality represents depth information in the form of depth maps. Depth adds an additional dimension to visual data that cannot be observed in RGB. Depth maps are 2D grayscale images that are the same size as their respective RGB images. Each pixel in the depth map has a gray level that indicates the distance between the camera and the corresponding RGB pixel. The depth map has each pixel set the position in the Z-axis of the corresponding RGB pixel. The depth pixels have values ranging from zero which corresponds to the furthest points from the camera, to 255 which corresponds to the nearest points from the camera (Sebai, 2024). Depth maps provide a lifelike stereoscopic display to users. There are two key features that separate depth maps from RGB images, the piecewise planar definitions, and the impact of depth discontinuities on

synthesized views. Synthesized views are reconstructions of the images or videos that have new viewpoints. Each plane represents an object in the scene where pixel intensities of the coplanar pixels vary smoothy. Contours are placed at the edges of objects, revealing a sharp depth discontinuity between objects in the foreground and the background of the scene.

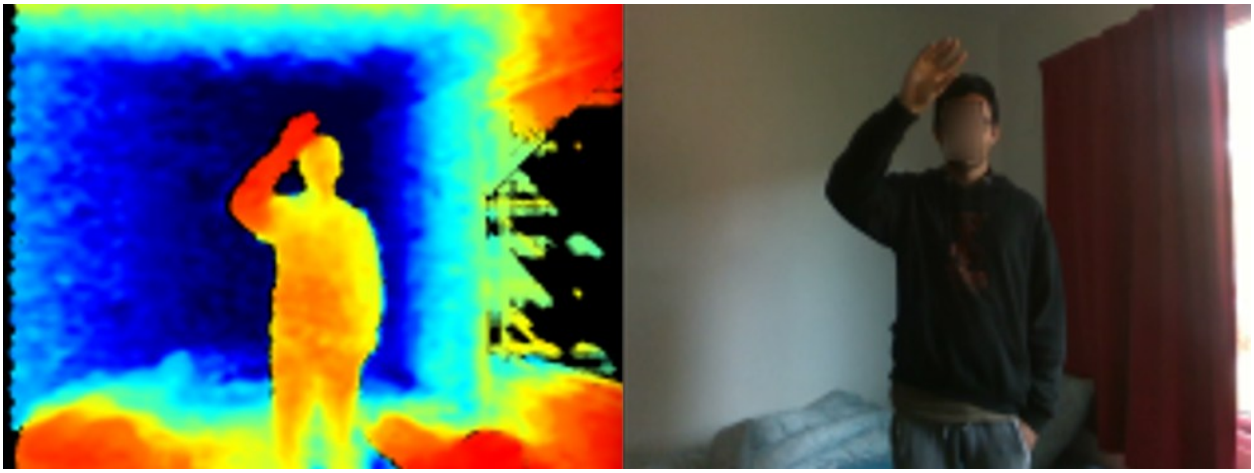

Figure 2.2 Same frame capture by depth (left) and RGB camera (right) (Jeeru et al., 2022).

2.3.2 Landmark Coordinates

Traditionally, to build a robust gesture recognition system, hand detection and tracking must be performed before feature extraction and classification. During hand and skin color segmentation, Kalman filters, background subtraction, edge detection, contour detection, and optical flow tracking are used (Tripathi and Verma, 2023). For spatio-temporal feature extraction, a histogram of oriented gradients, optical flow, scale-invariant feature extraction, geometrical feature extraction, and principal component analysis are used. Skeleton representation is compact and can robustly separate the action subject from the background.

In Zhang et al. (2020), MediaPipe Hands was introduced as an open-source pipeline to encourage the pursuit of gesture control and creative augmented reality (AR) and virtual reality (VR) applications. The architecture consists of two models that work together. First, the BlazePalm Detector, a single shot palm detector model which locates palms using oriented hand bounding boxes on a full input image. This is similar to BlazeFace which is also available in MediaPipe and used to detect faces which have higher contrasting patterns than hands. The lack of these contrast patterns in the hands makes them comparatively difficult to detect reliably just from the visual features alone. Therefore, a different strategy is used for BlazePalm. Instead of building a hand

detector, first a palm detector must be built and trained. This is due to the relative simplicity of estimating bounding boxes for dense and rigid objects such as a palm or fist compared to fingers. Additionally, the non-maximum suppression algorithm works well for two-hand self-occlusion cases like handshakes due to the relatively smaller palm objects. Palms can also be modelled only using square bounding boxes, ignoring other aspect ratios. After training the palm detector, an encoder-decoder feature extractor is used for larger scene-context awareness including smaller objects. Lastly, the focal loss is minimized during training to support large amounts of anchors due to the high scale variance.

The second stage in the MediaPipe pipeline architecture is the Hand Landmark Model. It is the hand landmarking model which operates on a hand bounding box which is provided from the palm detector, returning 21 keypoint landmarks through regression. The model learns consistent hand pose representation with the robustness to detect partially visible hands and self-occlusions (Samaan et al., 2022). The 2D coordinates are learned from real-world images and synthetic datasets with the relative depth with respect to the wrist point are learned through the synthetic images. Another output of the model was developed to recover from tracking failure. This output produces the probability of the event that a reasonably aligned hand is present in the crop provided by BlazePalm. If this output score is below a certain threshold, then the detector is triggered to reset the hand tracking. The binary classification head was developed to predict whether a hand in the input image is the right or left hand.

The hand tracking pipeline in MediaPipe is built with modular components named Calculators. These calculators come with MediaPipe as a set of tools to solve tasks such as model inference, media processing, and data transformations across a wide range of devices and platforms. The graph for the hand tracking from MediaPipe is shown in Figure 2.3 where there is a subgraph for hand detection and another for landmark computation (Andrushchenko et al., 2024). The palm detector only runs when needed, saving computational costs done by deriving the hand location from the landmarks in the previous frame and eliminating the need for the palm detector in each frame. The output from the hand tracking model is an additional scalar which captures the confidence that a hand is present and reasonably aligned in the cropped input image. If the confidence drops below a certain threshold, the hand detection model is reapplied to the next frame.

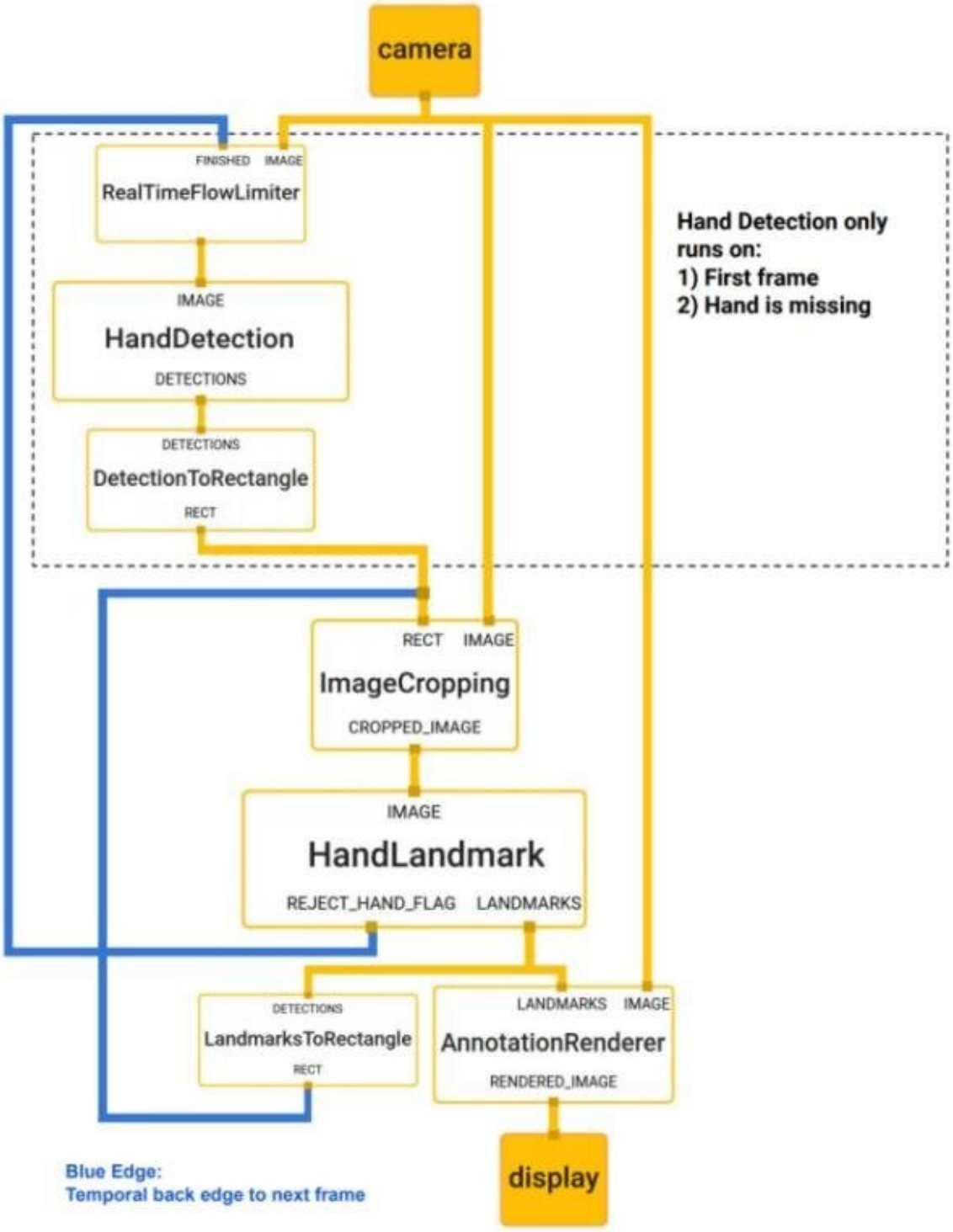

Figure 2.3 MediaPipe hand tracking pipeline subgraph (Andrushchenko et al., 2024).

2.3.3 Multi-modal Fusion

Mahmud et al. (2023) used multi-modal-fusion Convolutional Recurrent Neural Networks (CRNNs) on depth images and the corresponding 2D hand skeleton coordinates. Their approach can be described as multi-modal as they use skeleton coordinates in conjunction with quantized depth images. However, the raw depth images possess very low contrast in the hand region of interest (ROI). Fine details such as finger orientation, fingers overlapping with each other and the palm are difficult to distinguish. To address this, they proposed generating quantized depth images as an alternative input modality to the raw depth images, which increases sharp relative contrasts between the key parts of the hand and which leads to improved gesture recognition performance. They obtained accuracies of 93.81 and 90.24% on the SHREC-2017 dataset, significantly improving on the previous multi-modal-fusion CRNNs. However, ultimately their method does not outperform unimodal methods like DG-STA and PointLSTM.

In Balaji and Ranjan Prusty (2024) complementary depth-map and surface-normal vectors are fused through a cross-attention module to yield a multi-modal fusion hierarchical self-attention network. This paper's aim was to explore the addition of multi-modal data to a skeleton-based Hierarchal Self-Attention Network (MF-HAN) and test for increased model effectiveness. Unlike traditional fusion techniques, here the network uses features derived from other sources of multi-modal data in a reduced feature space using a cross-attention layer. Each frame in the video sequences contains a depth map with resolution 640x480 pixels denoted by $D \in R^{640\times480}$. By determining the normalized gradients across horizontal and vertical directions, the surface normals map can be calculated from D such that $\overline{N} \in R^{640\times480\times3}$. The horizontal and vertical normalized gradients, d_x and d_y are calculated by convolving the Sobel's kernels on the depth map D, with the surface normal vectors map considered to be $N = (d_x, d_y, 1)$. The surface normals map $\overline{N}$ is then simply calculated by normalizing N. The depth and normal maps are passed into the pixel encoder to generate embedding to match the hand gesture which is represented. The pixel encoder network works on the combination of the depth and normal maps for each gesture by splitting the maps into patches shaped 10x10x4 each, and tokenizing each with a pixel tokenization algorithm which is used to discriminate features in hand images. When there are multiple modalities available to be used as an input, a convolution layer with a kernel size of 3 with stride and padding of 1 is used to reduce the number of input feature channels to 1. Cross-attention in the decoder

architecture uses different feature vectors to generate queries, keys, and values in the attention module to facilitate the fusion of one modality with a different modality. The embeddings, once added to the baseline Hierarchical Attention Network (HAN) model, enable the model to incorporate skeleton features and image features together to create a rich representation of each gesture.

Modalities play a big part in gesture recognition systems that use vision-based cameras. There are many different modalities that can be used such as RGB and depth, coordinate systems, and even combinations of multiple different modalities. Nonetheless, when training a deep learning model for video gesture recognition, the modality must remain consistent throughout the dataset being inputted as seen in the following section.

2.4 Deep Learning Architecture for Video

Machine Learning is a widely used tool in Robotics, with applications such as reinforcement learning for robotic grasping (Cruz and Igarashi, 2020), or neural networks for predictive maintenance in robots (Koca et al., 2020). The two primary approaches for artificial intelligence and machine learning are supervised learning and unsupervised learning. The main differentiation between the two approaches is that supervised learning uses a labeled dataset while unsupervised learning uses an unlabeled dataset (Almuqati et al., 2024).

With supervised learning, algorithms are trained using labeled data that have correct corresponding outputs for the input data. The algorithms are made to classify new data or make predictions based on labeled data and patterns learned in training. Unsupervised learning instead focuses on uncovering patterns, structures, or relationships within unlabeled data without any specific guidance or instruction such as the labeled data needed for supervised learning. Similarities and differences in data positions can be identified through unsupervised learning, making it a great approach for tasks requiring exploratory data analysis and image recognition.

There are several common supervised and unsupervised learning algorithms. In supervised learning there is linear regression, logistic regression, decision trees, support vector machines (SVMs), and naive Bayes, all of which are approaches either for regression, classification, or sometimes both. In unsupervised learning there are two kinds of algorithms; clustering and

association, which encompass algorithms such as k-means, hierarchical clustering, and density-based spatial clustering of applications with noise (DBSCAN).

2.4.1 Convolutional Networks

Convolutional Neural Networks use spatial filters to learn visual patterns, and they have become a very common approach to image classification. There are three primary types of layers in a standard 2D CNN: 1) Convolution layers which apply a bank of small learnable kernels across an image to produce feature maps to capture edges, textures and shapes, 2) Pooling layers that down-sample the feature maps commonly through max pooling or average pooling to aggregate activations and build invariance to small translations, and 3) Fully connected layers that flatten and combine the feature activations into global representations which are used for final classification through a SoftMax output (Sahoo et al., 2022). There are several common CNN architectures such as VGG-16/19 which improved accuracy through deep stacks of uniform 3x3 convolutions Inception from GoogLeNet factorized convolutions into parallel branches of different kernel sizes, reducing computational cost. ResNet used identity shortcuts to enable training extremely deep networks up to hundreds of layers without vanishing gradients (He et al., 2016).

Extending from the traditional 2D CNNs, 3D CNNs introduced a temporal dimension to inputs and convolution filters, enabling the model to learn spatio-temporal features from video data (Naik and Soni, 2021). In 3D CNNs, a 3D kernel is applied over the input volume of a shape, with the added dimension representing the temporal depth or the number of frames. A typical CNN block alternates between convolution and pooling layers. The 3D convolution layer has stride and padding to learn low-level spatio-temporal features like motion edges and dynamics. The 3D pooling layer commonly uses max pooling and reduces the spatial and temporal resolution of the volume, limiting overfitting and computational cost. After a sequence of 3D convolution and 3D pooling blocks, feature maps are flattened into a 1D vector in the fully connected classification layers. Each dense layer outputs class probabilities. The addition of the third dimension to CNNs allowed models to learn motion features end-to-end more easily, improved accuracy in spatio-temporal correlations, and simplified the architecture by combining spatial and temporal procession into one process.

A temporal convolution network (TCN) is created by stacking units of one-dimensional convolution across the temporal domain leading to a non-linear activation function and max pooling. TCNs take a temporal sequence of *D*-dimensional feature vectors extracted per video frame as their input. For a video with T frames, the input X is a concatenation of all the D-dimensional feature vectors across time such that $X \in \mathbb{R}^{T \times D}$(Hou et al., 2019).

2.4.2 Hybrid Approaches

In Hax et al. (2024), they propose a hybrid architecture consisting of a Recurrent Neural Network including a long short-term memory layer, on top of a convolutional neural network, to recognize RGB dynamic hand gestures recorded in realistic environments. They used only six dynamic hand gestures: scroll left/right/up/down, zoom in/out. The implemented InceptionV3 model extracts the features and provides wrapped frame-feature maps as the input for the RNN which performs the final classification. The inputs for the RNN are the 2D spatiotemporal feature maps. The proposed model has an average accuracy of 83.7% with most of the misclassifications occurring when the model has to distinguish the scroll-right gesture from the scroll-up gesture, and the zoom-in gesture from the zoom-out gesture. The authors claim this is likely due to the unclear direction of movement before the gesture begins, and the similarities between gestures which causes the model to struggle with their generalizations. The hybrid approach is shown in Figure 2.4 where the 2D CNN extracts the spatial information from individual images, determining hidden patterns rather than performing classification. The feature maps for each frame in a sequence are passed to the LSTM network where the features and hidden patterns are extracted, observing temporal relations and then performing classification (Yaseen et al., 2024).

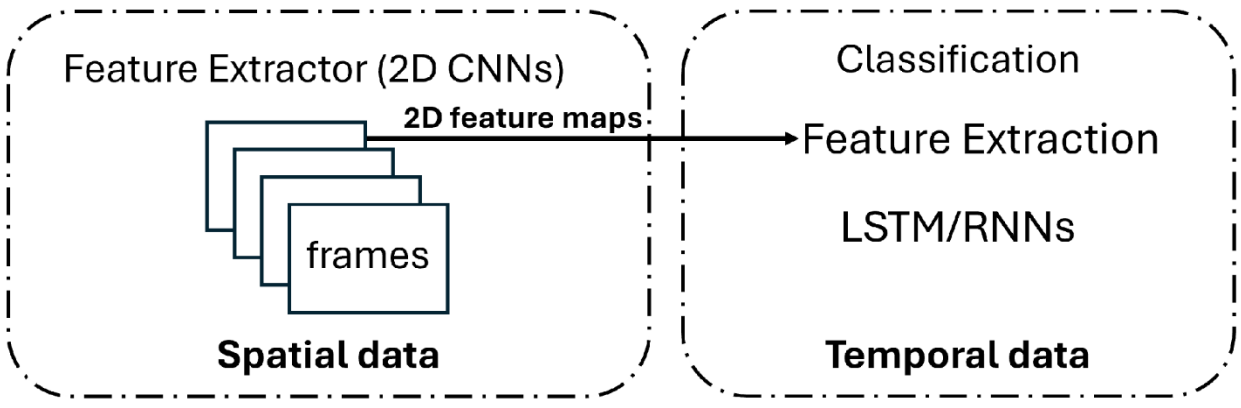

Figure 2.4 Feature extraction to LSTM-RNN (Yaseen et al.*, 2024).*

In Yaseen et al. (2024), they also used the “Depth_Camera_Dataset”. Their architecture consisted of three main blocks: MediaPipe to locate hands and extract region of interest (ROI), cutting the unwanted features from the frames, reducing dimensionality to (10, 50, 60, 3). InceptionV3 is next with a 2D CNN feature extractor with an output of a 1D row vector of size (10, 2048, 1). The data frame is then inputted into the LSTM block for final feature extraction and classification. Their model achieved an average test accuracy of more than 89.7%.

In Rastgoo et al. (2024), they observed Zero-Shot Learning (ZSL) as a solution to the annotation bottleneck in DHGR, when there are no annotated visuals, by leveraging their textual descriptions. They used an Auto-Encoder (AE) on top of an LSTM to make a trade-off between the dimensionality of the skeleton-based and deep features. A Bidirectional Encoder Representations from Transformers (BERT) model is used to map the visual features to the lingual embedding of the class labels. They achieved accuracies of around 60.2%-74.6% based on different large-scale datasets. The skeleton-based features included the distances and angles, and the singular values from the singular value decomposition (SVD) method were fused. The pixel-level features were obtained from the C3D model and were the inputs to the LSTM network. The AE on top of the LSTM balances the dimensionality of the skeleton-based features by reducing the dimension of the features from the LSTM’s output. The fused features were used in semantic space to map the visual features to the lingual embedding received from the BERT model. The authors proposed advancing the work by substituting the CNN model with the Transformer model.

2.4.3 Transformers and Attention-based Approaches

Many of the previous works focus on the utilization of convolutional neural networks and hybrid architectures that include them; however, in recent times there has been increasing investigation of the use of transformers. Garg et al. (2024) use the NVGesture NVIDIA Dynamic hand Gesture Dataset (Molchanov et al., 2016) and the Briareo dataset (Manganaro, 2019), with the Briareo dataset performing particularly well. This dataset contains dynamic hand gestures collected using RGB, depth and additionally an infrared camera *Pico Flex*. The dataset contains 12 different dynamic gestures, performed by 40 subjects, of which 33 were males and 7 were females. They implemented a model on Nvidia GeForce GTX 1080 Ti 12 GB GPU, CUDA 10.2 with cuDNN 8.1.1 and Torch 1.7.1 with the Adam optimizer. The model was trained on 40-frame sequences employing a batch size of 8, and a learning rate of $1e^{-4}$. They attempt to fuse combinations of 5

different modalities. The best accuracy from double modality fusion was 82.78% from normal and depth fusion. When adding a third modality, the fusion of infrared, RGB, and depth provided the best accuracy at 84.2%. Finally, the accuracy improved to its best accuracy with 5 modalities combined, as in color, depth, infrared, normal, and optical flow with the best overall accuracy being 85.85%. Their paper's main feature was the multiscale transformer. First, they used ResNet-18 to get the feature map for the complete hand gesture sequence input by concatenating all of the frames in the sequence. Spatial embedding is used to encode the extracted features of the complete sequence. Position and class token embedding are then added to the encoded features which are then inputted into the Multi-Video Transformer Network (MVTN). The transformer is adapted to extracting multiscale attention features so the model can learn the variations in hand shape and size efficiently. Sine-cosine position embedding is added to each frame before it is fed to the transformer so that the model knows the order of the sequence. Lastly, the classification head obtains the probability distribution.

Balaji and Ranjan Prusty (2024) propose moving from traditional convolutional architectures to attention-based networks. These networks have been proven to outclass CNN + LSTM (Long Short-Term Memory) architectures, particularly with skeleton-based attention networks. Transfer learning can be explored further by using pre-trained Vision Transformers (ViTs) or ResNet models. Different kinds of datasets with different modalities may also be analyzed when using ViTs or ResNet, such as RGB, skeleton, optical flow, infra-red, or any combination of them.

2.5 Datasets and Data Augmentation

2.5.1 Datasets

There are a handful of datasets that have been constructed to drive progress in vision-based dynamic hand gesture recognition. These datasets include several modalities such as RGB, depth, skeleton, and offer a variety in sizes, gesture vocabulary, and background conditions. The DHG14/28 dataset contains depth and skeleton data of the hand joints (De Smelt et al., 2016). It consists of 2800 gesture sequences amongst fourteen hand gesture classes, performed five times each by twenty participants. The gestures are performed both with one finger and with two fingers. The skeleton data contains twenty-two hand joint points to track the typical human hand. The dataset can be used for 2D and 3D hand shapes and used the Intel RealSense short-range depth

camera to collect the data. The Sheffield Kinect Gesture (SKIG) dataset combines 1080 RGB and 1080 depth sequences with 10 hand gesture classes, each performed 3 times by 6 different people, under 2 different illuminations with 3 different hand shapes (Tripathi and Verma, 2023). NVIDIA created a dataset with the SoftKinetic DS325 sensor, capturing RGB and depth data in a vehicle simulator, containing 1532 gestures performed by 20 participants spanning eighteen gesture classes (Molchanov et al., 2016). The SHREC 2017 Track dataset is similar to the DHG14/28 dataset, containing the same gesture classes and using a similar method for data collection. There are 2800 instances of data available in total in the SHREC 2017 Track dataset (Mahmud et al., 2023).

The Intel RealSense depth camera-based dataset of hand gestures was created for developing novel machine learning algorithms that efficiently classify and recognize different hand gesture videos. This dataset is the one used to train, validate, and test our model. In Hax et al. (2024) they only used the RGB frames, omitting the depth frames captured in the dataset. Averaging across five folds, their training accuracy was 96.73% with a validation accuracy of 84.7% and test accuracy of 83.7%. Their best single-fold test accuracy was 86.33%. They noticed that the scroll down gesture had the highest recall with the zooming out gesture having the lowest recall. Major misclassifications occurred between the scroll right and scroll up gesture, as well as the zoom in and zoom out gestures. In Leon et al. (2022) they use the depth camera dataset to develop a video-based hand gesture recognition using a lightweight CNN. The CNN was able to accurately classify video gestures with a limited number of frames. They reported a 99.84% accuracy on the RGB test data and 99.18% on the depth test data.

2.5.2 Synthetic Video Generation

In Chatterjee et al. (2024), Generative Adversarial Networks (GANs) are used in combination with convolutional neural networks to recognize static hand gestures through a camera. The GAN is used to generate images of hand gestures that expand to all possible variations while maintaining a close resemblance to the real ones. The CNN thus uses an expanded dataset which has the original real hand gesture images along with the generated images. The pretrained ImageNet CNN architecture was a modified VGG16 architecture with an embedded self-attention module that allows the model to focus on distinguishing features to improve the overall differentiability among the hand gestures. The GAN utilizes adversarial training described as a two-step process which

first trains the discriminator on real and synthetic hand gesture images, then the generator attempts to produce synthetic images that may deceive the discriminator. This training loop leads to iterations which can refine both the discriminator and the generator. Although use of the GAN is a recent advancement in HGR, there is still room for improvement due to the variability and complexity of real-world situations. The model would benefit from more diversity in hand shapes, sizes, and environmental conditions in the dataset to accommodate individuals with varying ages, genders, cultural backgrounds, and abilities.

In Csonka et al. (2023), to increase performance, the image dataset was enriched using data augmentation techniques with gestures that included rotations, horizontal flips, zooms, and translations. Additionally, a Python data generator function was used to flip each image horizontally, and resize to 250 pixels in height and width, along with the pixel values normalized to the range [0, 1]. The data generator algorithm aids in optimization of efficiency as it reads the data in batches, processes them, and forwards them for training, saving memory while facilitating extensive datasets in training. This is a common method widely used in the fields of computer vision and machine learning.

There have been several datasets put together for hand gesture recognition applications. These datasets include different modalities and structures, along with different gestures and poses. As many of these datasets are not suitable to be used together, there have been recent advancements in developing generative adversarial networks that can create new and useful additions to current and future datasets.

2.6 Robotic Control Using HGR

The Franka Emika Robot, which provides transparent 1-kHz interfacing with accurate kinematics and dynamics models, was the first commercially available, fully safety-certified tactile robot (Haddadin, 2024). This robot can automate previously nonautomated tasks such as assembly, machine tending, gluing and welding, packaging, palletizing, and quality testing, through easily programmable robotic system integration compared to traditional methods (Zinser et al., 2025). With artificial intelligence and machine learning making big advances, the demand for data interfaces and high-accuracy state measurements has increased for robots with integrated joint torque control. The Franka Emika Robot is designed for artificial intelligence and software

development, built upon three core principles. It should be capable, affordable and accessible, its dynamics should reflect rigid-body Lagrangian robot behavior from an input-to-state perspective, and it should support high performance 1-kHz real-time control of the desired joint torques in addition to other motion commands such as Cartesian velocity. A simulation of this robot with its labeled joints is shown in Figure 2.5.

In Gallouédec et al. (2021), the Franka Emika robot's Panda robot arm was simulated to evaluate a reinforcement learning algorithm for complex manipulation tasks, same as the simulated robot in Figure 2.5. The robot has 7 degrees of freedom and a parallel finger gripper. The open-source PyBullet physics engine (Coumans and Bai, 2016) was used to simulate the robot. The authors describe panda-gym, a free and open-source package that allows robotic tasks to be defined. The tasks consisted of moving either the gripper or one or more objects to a target position, with the task being completed when the distance between the gripper or object and the target position is less than 5 cm.

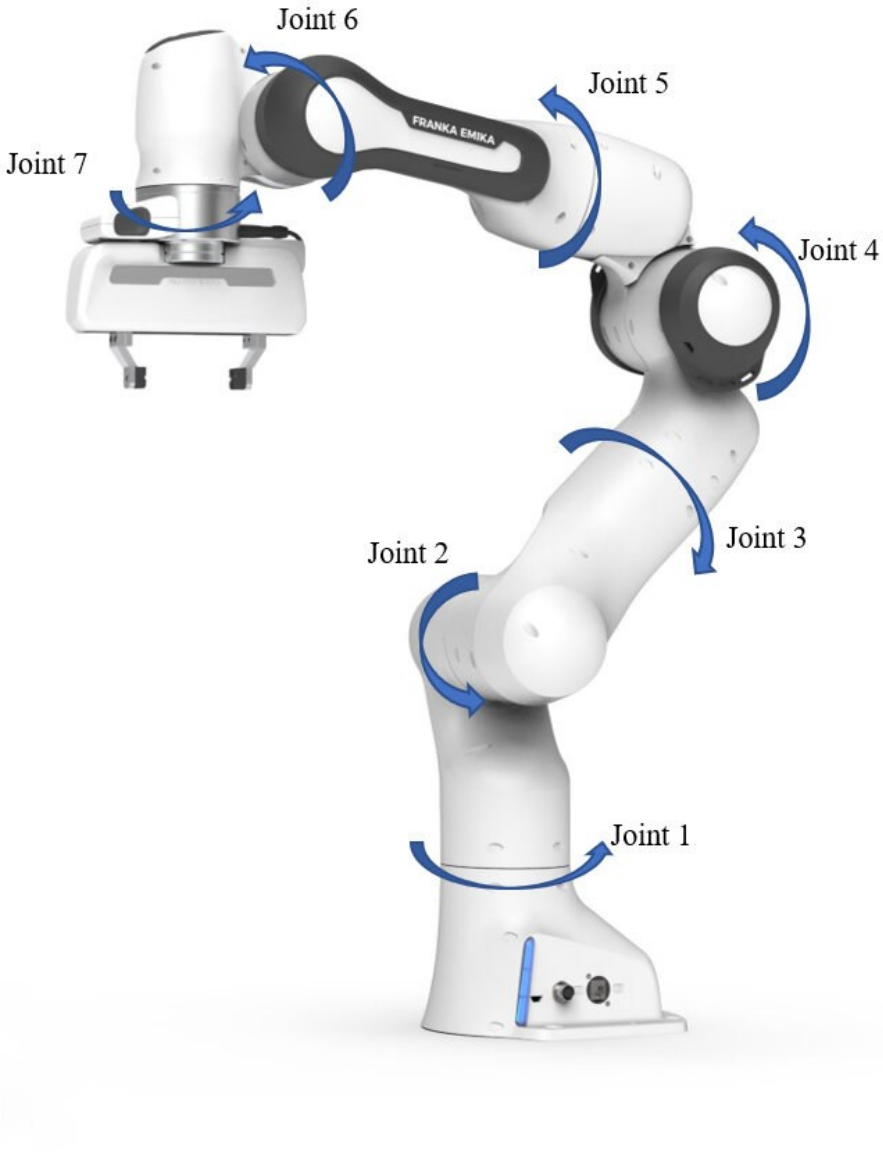

Figure 2.5 Simulated Franka Emika Panda 7 DOF Robot with labeled joints (Rogel et al., 2022).

A low-latency system for gesture-based teleoperation by combining 3D landmark extraction with continuous gesture-derived mobile control is proposed in Xie et al. (2025). Their work demonstrates an end-to-end real-world teleoperation pipeline where a depth-MediaPipe framework extracts 3D coordinates of the 21 hand keypoint landmarks, and a Semantic-Pose to Motion (SPM) model is developed to interpret the poses and semantics of hand gestures. The model translates the 3D coordinates into real-time mechanical actions used in locomotion, robotic arm end effectors, and quadruped robots. An Intel RealSense D435i camera is used to capture RGB images and depth maps together, with the keypoints extracted from the RGB images and mapped to the depth images. The system can infer command switching from the SPM; however dynamic video-based hand gestures are not included, and instead the switching between static hand gestures is recognized.

Based on the review of past studies, we can see that there are gaps that motivate this work. Previous work relied on only the RGB modality of the dataset, not making use of the depth modality. Public datasets are also typically small, with limited modalities, low diversity in subjects, lighting, camera viewpoint, speeds, and no substantial robotic control aspects. Therefore, there are opportunities to fuse modalities while preserving full sequences, and implement robotic control for end-to-end HRI applications.

3. Methodology

This chapter presents the different components of the system that was developed. Section 3.1 describes the dataset that was used in the experiment. Section 3.2 explores the four main software elements of the system, TensorFlow in Section 3.2.1, OpenCV in Section 3.2.2, Scikit-Learn in Section 3.2.3, and Matplotlib in Section 3.2.4. Section 3.3 presents the classification module developed for the system. Section 3.3.1 covers the preprocessing and how the dataset was split up, 3.3.2 describes feature extraction, 3.3.3 is on the learning algorithm's architecture, and 3.3.4 is on the training and validation procedures. Section 3.4 presents the testing and evaluation procedure for the experiment. Section 3.5 presents the computational resources used in the experimentation.

3.1 Dataset

The "Depth camera-based dataset of hand gestures" by Jeeru et al. (2022), available on the Mendeley Data website under the repository name "Depth_Camera_Dataset" was used in this research. The dataset was sourced from the ACPS group, Department of Information and Communication Technology at the University of Agder, Grimstad, Norway. It contains frames of videos capturing various hand movements in RGB and depth modalities using the Intel RealSense Depth Camera D435. The camera has a maximum range of 10 meters and provides two channels, one for capturing RGB streams and another for depth streams. The field of views (FOVs) for the camera are 87° x 58° for depth and 69° x 42° for RGB. The D435 camera is self-calibrated and supports multi-camera configuration with a hardware sync signal. The default self-calibrated settings were used for all measurements; however, the depth stream which is supposed to have a resolution of 480 x 860 was lowered to 480 x 640 to match the resolution of the RGB stream. The synchronization allowed for all the parameters to be synchronized across both the RGB and depth versions of the image. The frame rate of the camera at the highest resolution was between 30 and 60 frames per second, and at the lowest resolution, which was used for this dataset, it reached its highest frame rate of 90 frames per second. To maintain stability, the camera was mounted on a tripod stand when collecting data.

The dataset comprises 29,718 frames in total including both RGB and depth versions and is 10.6 GB in total size. These frames correspond to various hand gestures from various individuals, collected at different time instances and under complex background conditions. There are six hand

movements in the dataset including scroll-right, scroll-left, scroll-up, scroll-down, zoom-in, zoom-out. There are 662 recorded sequences for each gesture in each of the RGB and depth modalities, each sequence consisting of 40 frames. The primary objective of this large and diverse dataset was to accurately classify hand gestures in real-world scenario environments with complex backgrounds and diversified lighting conditions. The structure of the data repository is shown in Figure 3.1. The authors opted for a folder-based labeling approach. The root folder is divided into two subfolders: Depth and RGB. Each of those subfolders contains six subfolders, one for each of the six hand gestures and named appropriately. Each of the hand gesture folders contains 662 subfolders that represent videos captured at various times and backgrounds. Lastly, each of the 662 video folders contains forty image frames as .png files. For each of the gestures, the videos are diverse in subjects and times. The data collected consisted exclusively of different frames corresponding to the hand movements and contained no other personal information. In Jeeru et al. (2022) they state that the dataset was a free-for-all campaign and people came forward at their own discretion to provide their hand gestures. Examples of each of the gestures included in the dataset are shown in Figure 3.2, and a comparison between their RGB versions and depth versions are shown in Figure 3.3.

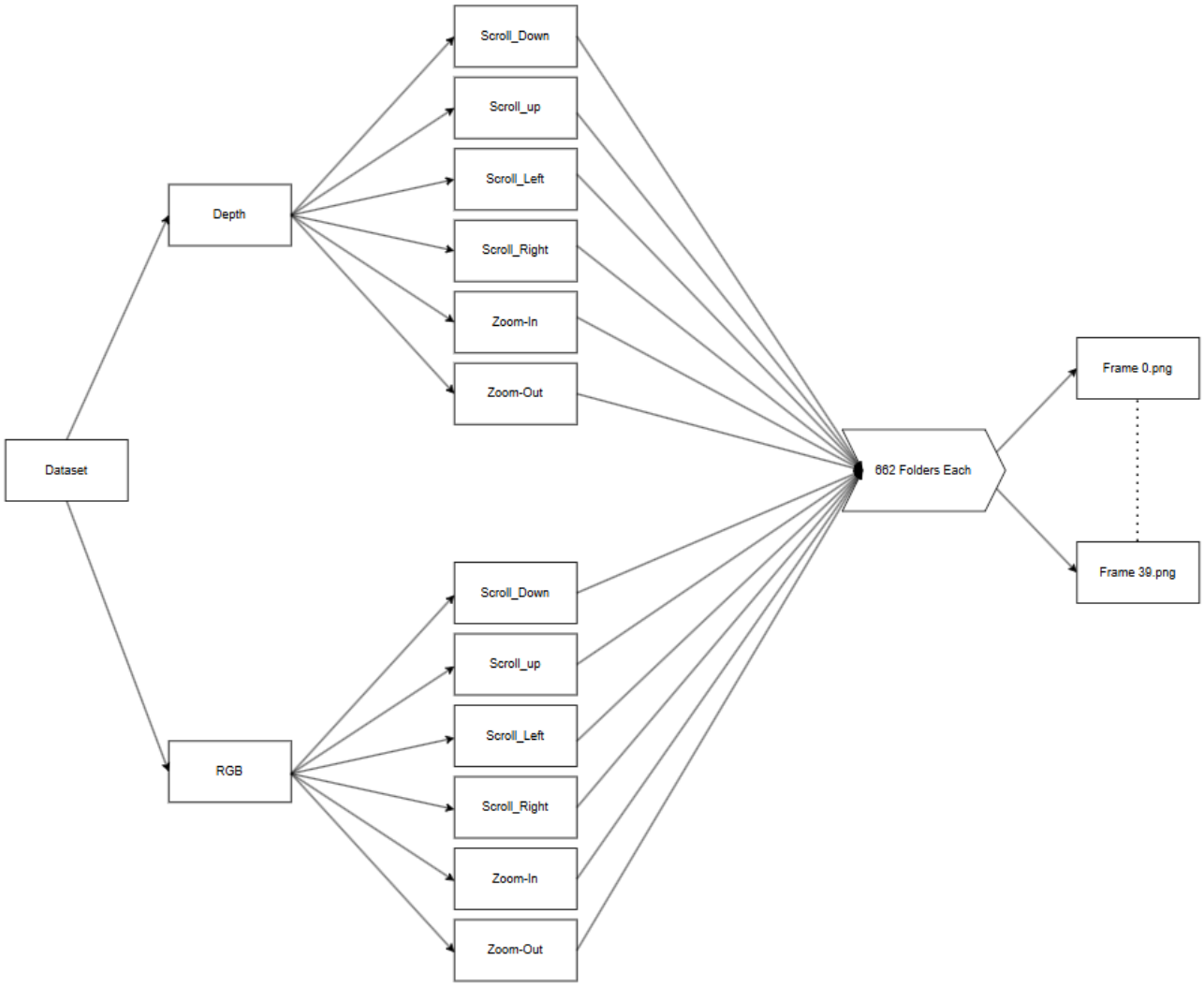

Figure 3.1 Flowchart of Depth_Camera_Dataset repository structure (Jeeru et al., 2022).

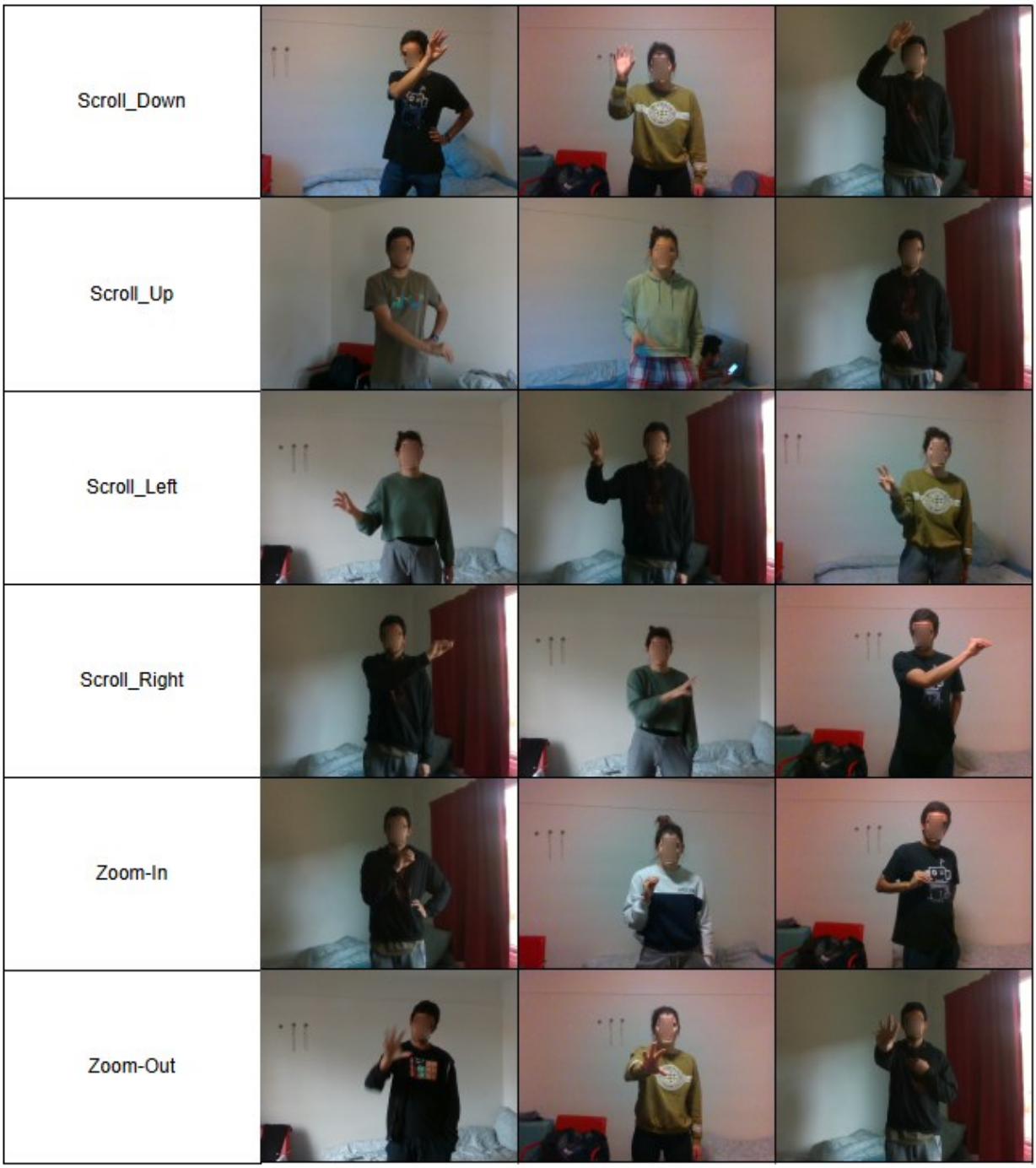

Figure 3.2 Example of diversity across different sequences in each gesture of the dataset.

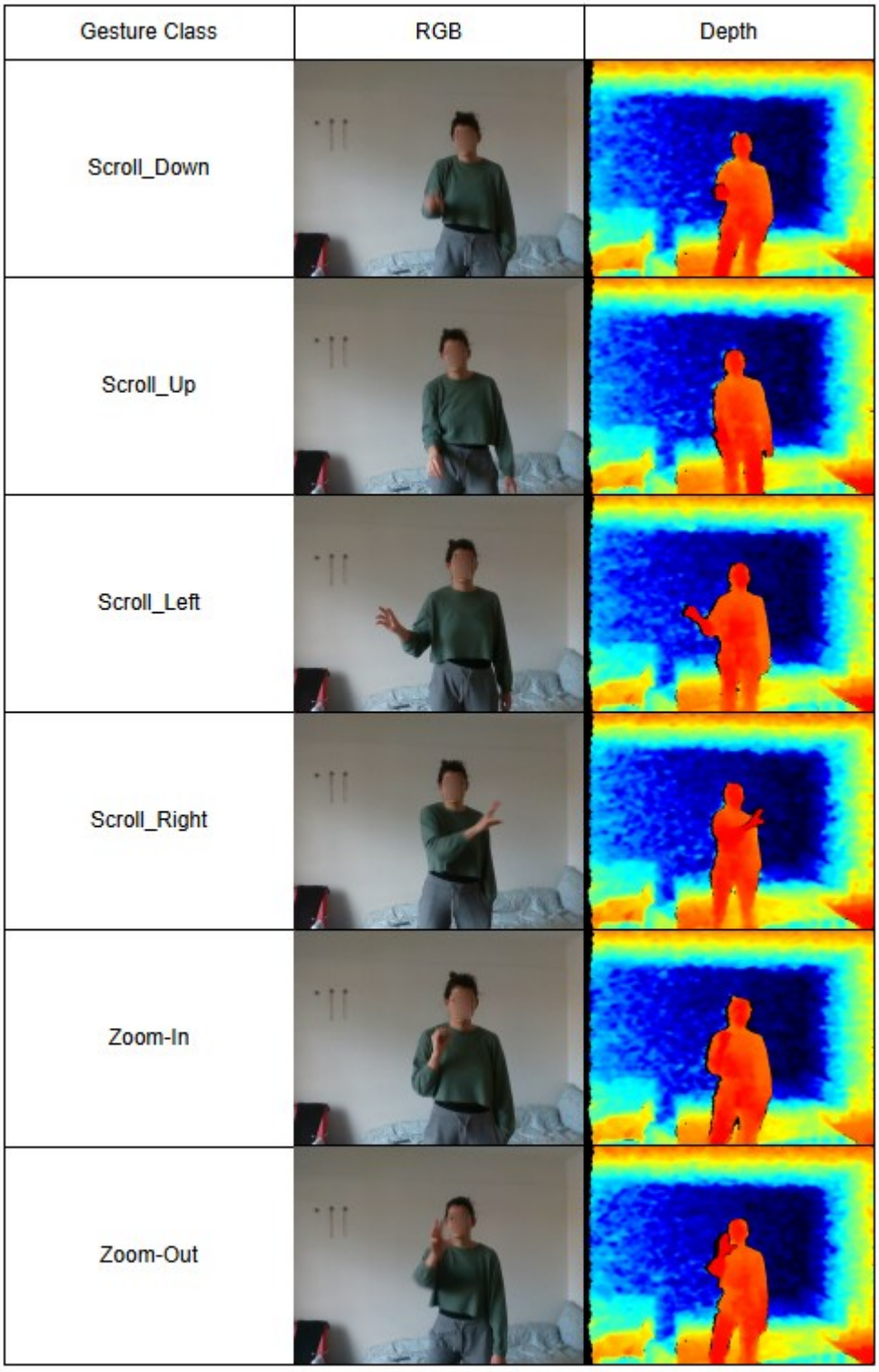

Figure 3.3 Examples comparing RGB and depth versions of each gesture in the dataset.

3.2 Software

3.2.1 TensorFlow

TensorFlow is an end-to-end platform for machine learning developed by Google Brain (TensorFlow Basics, 2024). It facilitates multidimensional array-based numeric computation, GPU and distributed processing, automatic differentiation, model construction, training, export, and more. Tensors are the multidimensional arrays which TensorFlow operates on; they are represented as tf.Tensor objects when programming. Large calculations tend to be slow when running on CPU, and TensorFlow is able to use GPUs to accelerate executing operations. In this project, a dedicated virtual environment was created using venv to isolate all the dependencies and allow easy reproducibility. The virtual environment includes TensorFlow, NumPy, along with supporting packages. There were several key APIs provided by TensorFlow's modular architecture including the Keras API for model construction, data API for input pipelines, and config API for device management and GPU control. GPU acceleration was configured using CUDA 12.2 and cuDNN 8.9.

Keras as a part of TensorFlow offers a higher-level API specifically designed for experimentation with neural networks and layers (TensorFlow: Keras, 2024). Keras' Sequential API is used to group a linear stack of layers to define a model's architecture. These layers include LSTM, Dense, Dropout, and TimeDistributed. Additionally, TensorFlow's ImageDataGenerator is used for data augmentation, applying random transformations to the dataset's images. Certain augmentations such as flipping and temporal augmentations had to be avoided since applying them to some gestures would make them indistinguishable from some of the other gestures. The augmentations that were applied remained consistent throughout each sequence, with all of the frames in that sequence having the same augmentation, but there being different augmentations from sequence to sequence. Grouping the layers together to make a model allows for straightforward compilation with optimizers, loss function definition, and tracking of evaluation metrics such as accuracy, precision, recall, F1-score, and Receiver Operating Characteristic Area Under the Curve (ROC-AUC).

TensorFlow's mixed precision uses both 16-bit and 32-bit floating-point types in a model during training to increase running speed and minimize memory usage (TensorFlow: Mixed precision, 2024). The model will have a lower step time by keeping certain parts of the model in the 32-bit types for numeric stability, while training equally as well in terms of the evaluation metrics. By setting the global policy to mixed_float16, the operations with higher computational cost are run in float16 to take advantage of the hardware units such as NVIDIA Tensor Cores while the variables and numerically sensitive parts remain in float32. TensorFlow enabled GPU acceleration and distributed strategy classes to parallelize the computational workload across multiple devices available from The Digital Research Alliance of Canada. The Keras API allowed simple experimentation with model architecture to allow quick iterations and hyperparameter optimization. Table 3.1 shows the versions of the TensorFlow libraries which were used in this work.

Table 3.1 TensorFlow Library versions used in this work

Library	StdEnv	gcc	cudacore	python	clang	cuda	cudnn
Version	2023	12.3	12.2.2	3.10.13	17.0.6	12.2	8.9

3.2.2 OpenCV

Open Source Computer Vision Library (OpenCV) is one of the largest open source computer vision and machine learning libraries in the world. Released under the Apache 2.0 license, it provides 2500 algorithms each optimized for a computer vision task. OpenCV is organized in modular components comprising 15 main modules and 60 extra modules.

Common applications are in image transformations, object detection, face recognition, and analysis of videos in real time. OpenCV's Python interface has core functions for reading and writing images as inputs or outputs, color conversions, and geometric transformations such as resizing or rotating. It is used to load and read images from the dataset, ensuring they are in the correct format of grayscale or color and checked for validity. It can resize images to enforce consistent inputs to the feature extractor, preparing the image as a NumPy array. These functions enable direct integration in machine learning libraries in Python.

3.2.3 Scikit-Learn

Scikit-Learn is another open-source Python library, providing tools for predictive data analysis. It was built on NumPy, SciPy, Matplotlib, and released under a Berkeley Software Distribution (BSD) license (Pedregosa et al., 2011). Scikit-Learn offers an API for a variety of supervised learning tasks such as classification, unsupervised learning such as clustering, preprocessing utilities such as feature scaling, model selection such as grid search, and performance metrics such as F1 score. There were several objects from Scikit-Learn that were used in this project.

- **StratifiedKFold**: Class-wise stratified k-Fold cross-validator. This object is a variation of KFold which returns stratified folds made from preserving the percentage of samples for each class in a binary or multiclass classification setting. The stratified 5-fold cross-validation used in this work is shown in Figure 3.4 where each of the 5 folds is used as the validation fold once while the other 4 are used for training. Each fold has instances for each of the six gestures.
- **GridSearchCV**: Exhaustive search over specified parameter values for an estimator. It implements a fit and score method along with score samples and predicts probability. Decision function, transform, and inverse transform can also be implemented. The parameters are optimized by cross-validated grid-search over the defined parameter grid.

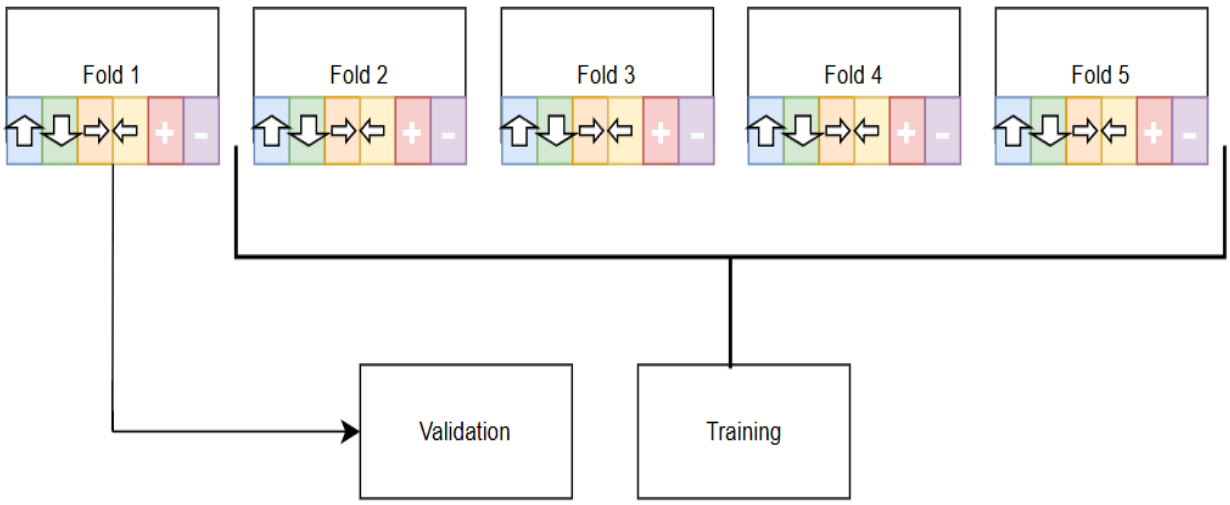

Figure 3.4 Stratified 5-Fold Cross-Validation grouping with the six symbols in each fold representing the gesture classes.

3.2.4 Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Designed for scientific and engineering visualizations in Python, Matplotlib offers a MATLAB-like pyplot interface with a more flexible object-oriented API that seamlessly integrates with NumPy arrays and Pandas DataFrames to render line plots, scatter plots, bar charts, histograms, pie charts, etc. Trajectories of simulated robots can be plotted as well, with 3D viewing angles that are adjustable.

Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing aesthetic and informative statistical graphics. It is able to help produce annotated heatmaps which can be used in normalized confusion matrices.

3.3 Classification Module for Dynamic Hand Gesture Recognition

3.3.1 Data Split

Learning the parameters of a prediction function and testing it on the same data would be a mistake in the methodology. A model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. The situation described refers to overfitting which can be avoided when performing supervised learning by holding out a part of the available dataset as a test set. Additionally, a further section of the dataset can be separated as a validation set which comes before the test set in the training procedure. To implement the validation set into training, k-fold cross-validation (CV) is used where the training set is split into *k* smaller sets. The final performance metric is the averaged score across all of the folds. In this project, the training set and testing set were separated, with the training set put through a stratified 5-fold cross-validation. GridSearchCV goes through combinations of defined model parameters and uses nested CV to identify the best estimator based on mean validation score. The model that was optimized through GridSearchCV was evaluated on the test set that was unseen and scikit-learn's accuracy, precision, recall scores, and confusion matrix functions were used to determine the predictive performance of the model.

The dataset was split into training and testing subsets to ensure the dynamic hand gesture recognition model had robust training and evaluation. A sequence-based approach was taken to split the data, preserving the temporal data within each gesture sequence (Figure 3.5). In contrast

to the Hax et al. (2024) and Yaseen et al. (2024) studies where from each sequence only every fourth frame was selected for use, this work used all 40 frames in each sequence rather than only 10 per sequence. Each RGB sequence in the dataset had a corresponding depth version. Care was taken to ensure that the two modalities of each gesture were placed in either the training or test set together. This approach ensures that the datasets maintain the temporal dynamics in the hand gestures and avoids leakage from sequences appearing in both training and test sets as different modalities.

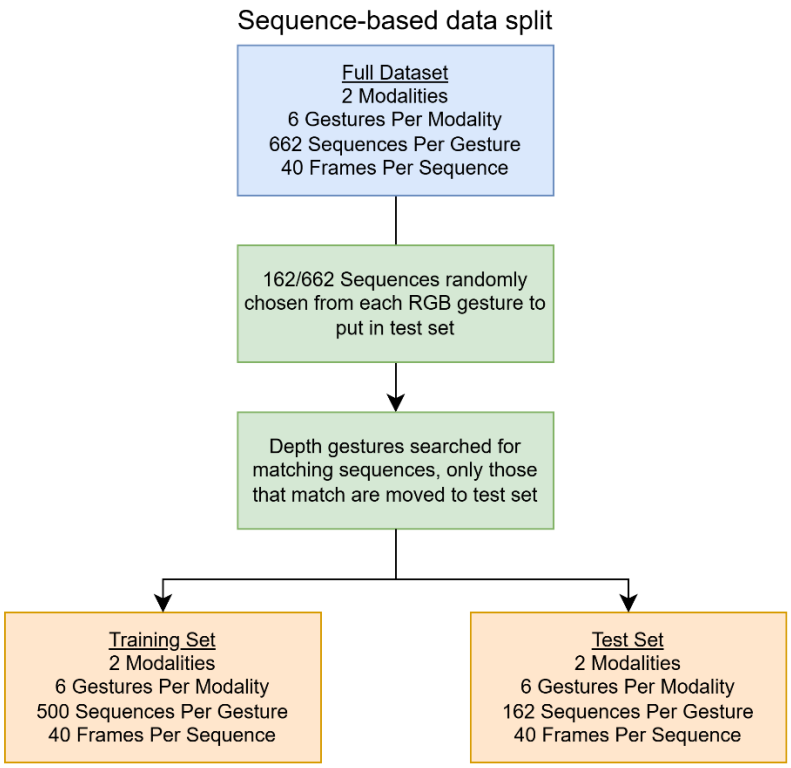

Figure 3.5 Flow chart of sequence-based data splitting.

The training set was used to train the model and contains the majority of the data. From each gesture class in each modality, 500 of the 662, or 75% instances were used for training and validation of the model.

The validation folds were used to monitor the model's performance and prevent overfitting during the model's training. Instead of selecting individual frames from each sequence, entire sequences

were selected and reserved for validation as seen in Figure 3.5, ensuring that the model was validated on different subsets in each iteration.

The testing set was used to evaluate the final model's performance. 162 full sequences from each gesture type were separated from the training set to create the testing set, accounting for 25% of the dataset. The data in the testing set was hidden from the model until it was time for testing. Each of the 5 folds from the training were saved as a Keras model and tested on the full test set, providing test results for 5 different folds. This ensured unbiased evaluation of the model's ability to recognize dynamic hand gestures in new and unseen data.

3.3.2 Feature Extraction

Before extracting the features, the following techniques were implemented to enhance the utility of the dataset for training the dynamic hand gesture recognition model:

- Normalization of the pixel values of all the RGB and depth frames by dividing by 255 to achieve the range of [0, 1], ensuring consistent input values for the neural network. Normalization makes sure that the model learns from the input images more effectively, maintaining numerical stability. This is in contrast to using the normalization function for InceptionV3 which divides by 127.5 to achieve a [-1, 1] normalization. Since in this work, all the normalization was consistently done to the range of [0, 1], the model is able to compensate and still works.
- Making sure all frames have a standardized dimension of 224x224 pixels, resulting in uniform input size for the model. The standardized dimensions of the frames are crucial for maintaining the aspect ratio across all of the images in the dataset and avoiding distorted hand gestures. However, it is important to note that the aspect ratio of the original images was not fully preserved.
- Color space conversion from the raw BGR data to RGB for compatibility with InceptionV3.

Pre-trained InceptionV3 models were used for feature extraction. Initialized with pre-trained weights from the ImageNet dataset which contains over a million images across a thousand classes, the model learned rich feature representations from this large and diverse dataset through transfer learning, leading to a strong foundation for image classification applications.

The feature extraction was part of the work done on the Graham cluster of The Digital Research Alliance of Canada. The weights of the InceptionV3 with no top were manually downloaded from their website as an h5 file. Feature extraction was performed using a pre-trained InceptionV3 model, where the top layers were replaced with a GlobalAveragePooling2D layer instead. Each sequence processed through the InceptionV3 model provided visual feature representations. These visual features are learned patterns that the neural network has discovered, and these can include hand position relative to the body, open palm versus pinched fingers, distance from camera, hand size, the angle of the arm's position, etc. The feature extraction performed in this pipeline can be referred to as offline feature extraction. This is when the CNN, or InceptionV3 in this case, is frozen, losing the end-to-end operation of the pipeline. Feature extraction was performed this way to speed up the preprocessing of the features after InceptionV3 had been set up and was working satisfactorily. Once the feature extraction is switched to offline mode, it only needs to be run once to compute the features, then those features which were extracted offline can be fed straight into the model. This offers a significant boost in speed and training stability in comparison to the embedded feature extraction which had to be run every time the model's training script was run.

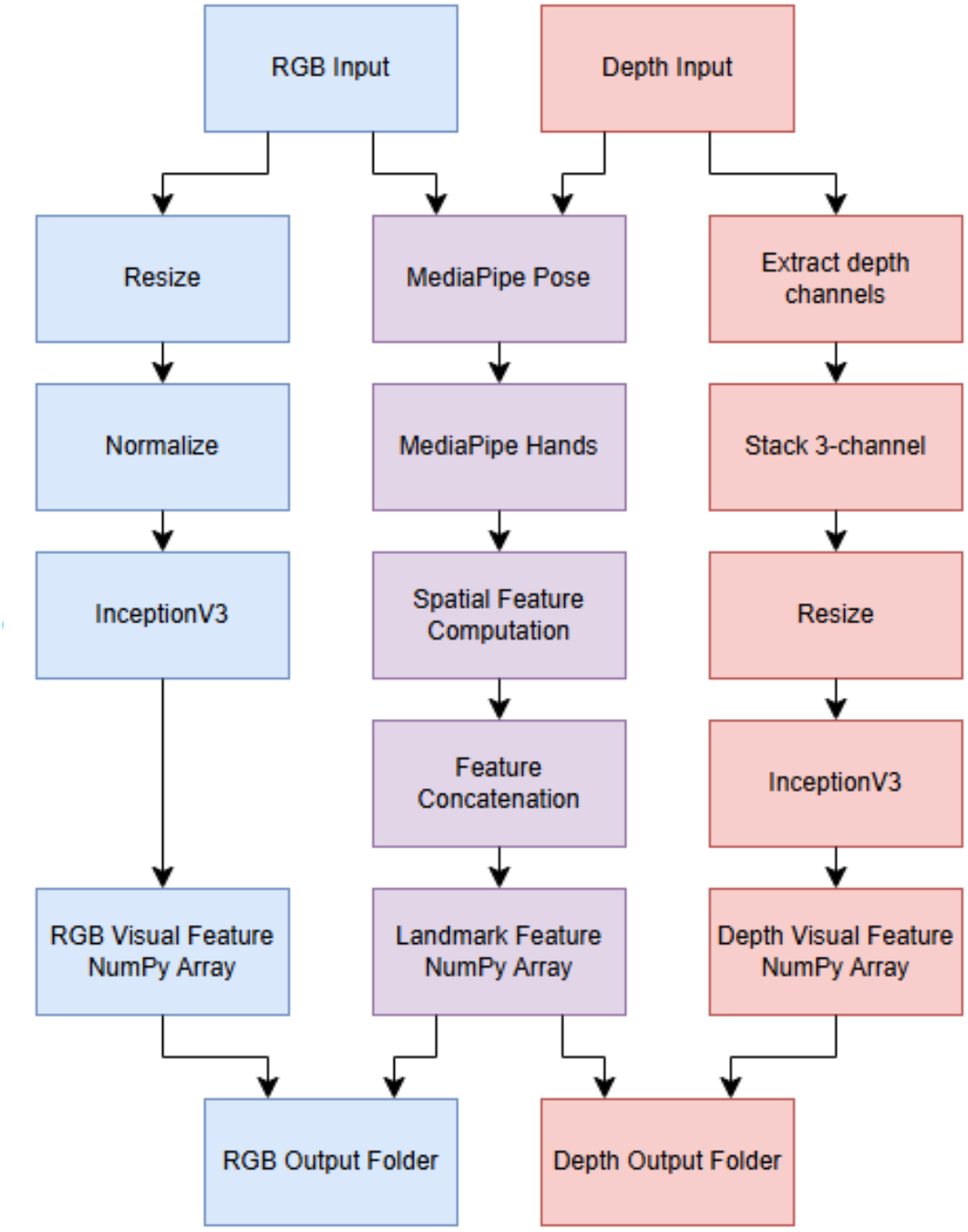

Figure 3.6 Parallel visual and landmark feature preprocessing pipeline flowchart. The operations shown are described in the text.

For each sequence there are two parallel feature extraction pipelines. These steps were crucial for preparing the data from the acquired dataset for training the model. The pipeline is shown in the flowchart in Figure 3.6. The landmark extraction pipeline extracts skeletal information from the sequence. The frames of each sequence are loaded using OpenCV and converted from BGR to

RBG color to be compatible with MediaPipe. The MediaPipe Pose extracts 33 pose landmarks as x, y, z coordinates while MediaPipe Hands extracts 21 hand landmarks. In addition, the hand bounding box area and average depth of hand landmarks are computed. The bounding box area is calculated by first detecting the hand landmark dedicated to the palm, and extending a scaled bounding box around it that is large enough to fit the hand inside. Due to the quality of the dataset, there was no case of MediaPipe missing the detection of a hand, but if this were to occur then the frame would be dropped. All extracted landmarks and spatial features are concatenated into a single NumPy array. The visual feature extraction pipeline handles the visual data for each of the RGB and depth sequences; however, each modality has its own pipeline as well due to the difference in the information contained in them. The depth data in the dataset was encoded as RGB24 format, so the depth was decoded by creating a stacked three-channel representation. The depth feature extraction pipeline is shown in Figure 3.7. Channel 1, pseudo-depth, contained the depth information based on a color mapping which had warmer colors as closer to the camera and colder colors as further from the camera. Channel 2, depth discontinuities, had edge information that indicated depth discontinuities. Channel 3, luminance, was the luma value which was computed based on the ITU-R BT.601 standard, where each color has a different weight with green being the highest due to the human eye being more sensitive to it than red or blue. Each of the RGB and depth pipelines has its own InceptionV3 model with its own weights so that the two modalities do not affect each other when the model is learning.

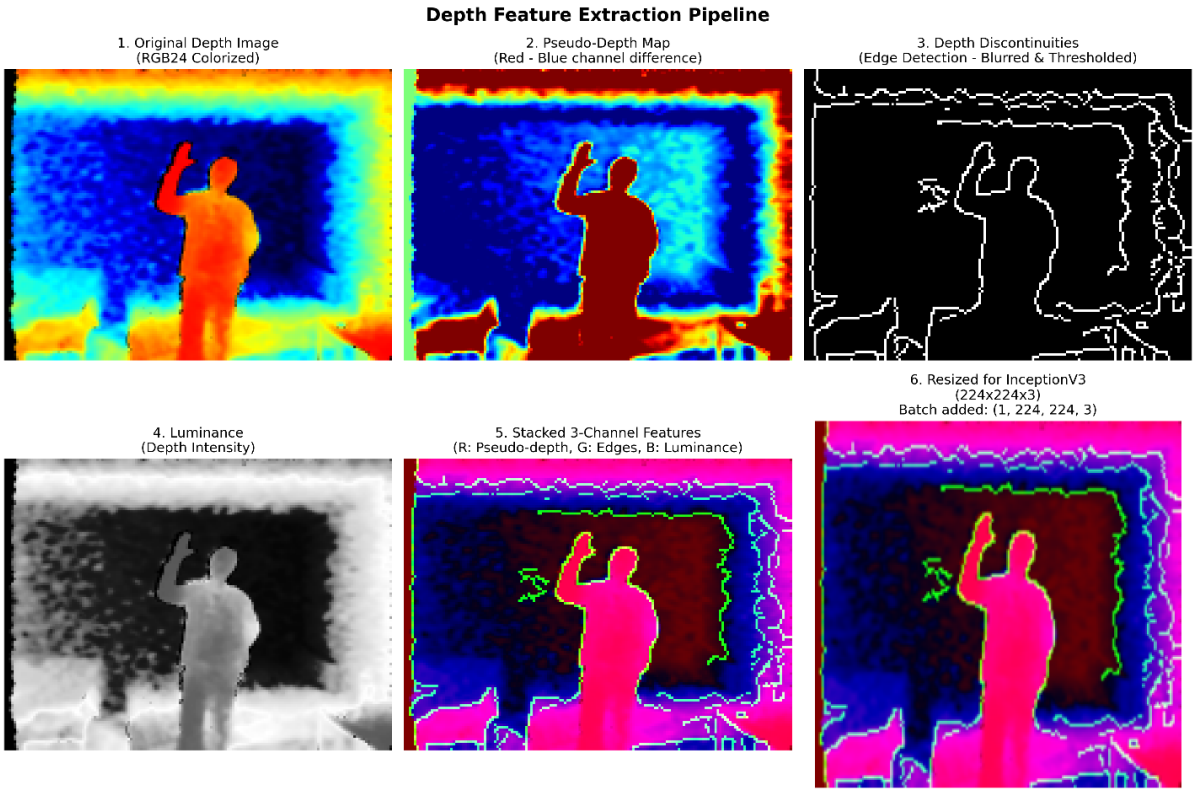

Figure 3.7 Depth feature extraction pipeline, pseudo-depth, depth discontinuities, and luminance.

The InceptionV3 model is a state-of-the-art convolutional neural network architecture with high performance in image classification tasks (Keras Documentation: InceptionV3, n.d). It is a part of the Inception family of models developed by Google and is commonly used due to its efficient and accurate feature extraction (Szegedy et al., 2016). InceptionV3 is the third version of Google's Inception CNN, introducing new procedures such as RMSProp Optimizer, factorized 7x7 convolutions, BatchNorm in the auxiliary classifiers, and label smoothening. Through the factorized 7x7 convolutions, the number of parameters can be decreased in a network without reducing its efficiency (Linkon et al., 2021). InceptionV3 has demonstrated effectiveness in several applications such as skin cancer classification to accurately identify different skin lesions. Additionally, it can be used to detect defects in plastic parts produced by injection molding, identifying short forming and weaving faults accurately, and has even been used in agriculture to identify early signs of banana diseases (Sanga et al., 2020) and blight disease in potato crops (Dutta et al., 2024), helping farmers improve their yield. The deep learning architecture of this version of Google's Inception CNN consists of multiple parallel convolution layers with varying kernel sizes and depths. These layers capture different features such as edges, textures, and shapes in input

images. The outputs of the layers are combined to make a prediction about the input image's class or category.

The InceptionV3 model must be adapted for any specific task, so the top layers which are responsible for the final classification in the ImageNet dataset were removed. The model's output was connected to a GlobalAveragePooling2D layer instead, which computed the average of all the feature maps. This resulted in a compact feature vector that captured all the essential characteristics of each input frame from each sequence. This approach of using a GlobalAveragePooling2D layer was inspired by the Hax et al. (2024) paper.

To extract high-level features, each frame of each sequence from the RGB and depth modalities had to be processed through the respective RGB or depth pipeline's InceptionV3 model. Each model processes the data and extracts 2048 deep features with the RGB model processing more standard normalized RGB images and the depth model processing the three-channel depth data. For each instance of each gesture, two NumPy arrays are created, one from the landmark pipeline and one from the visual pipeline.

For multi-modal fusion, the features are organized into four streams. RGB visual features, RGB landmark features, depth visual features, and depth landmark features. A four-branch neural network architecture is used to process and fuse the feature streams. The modalities are fused using the hierarchy seen in Figure 3.8. First the RGB visual features and RGB landmark features are concatenated, then the depth visual features and depth landmark features, and finally the RGB and depth modalities are concatenated.

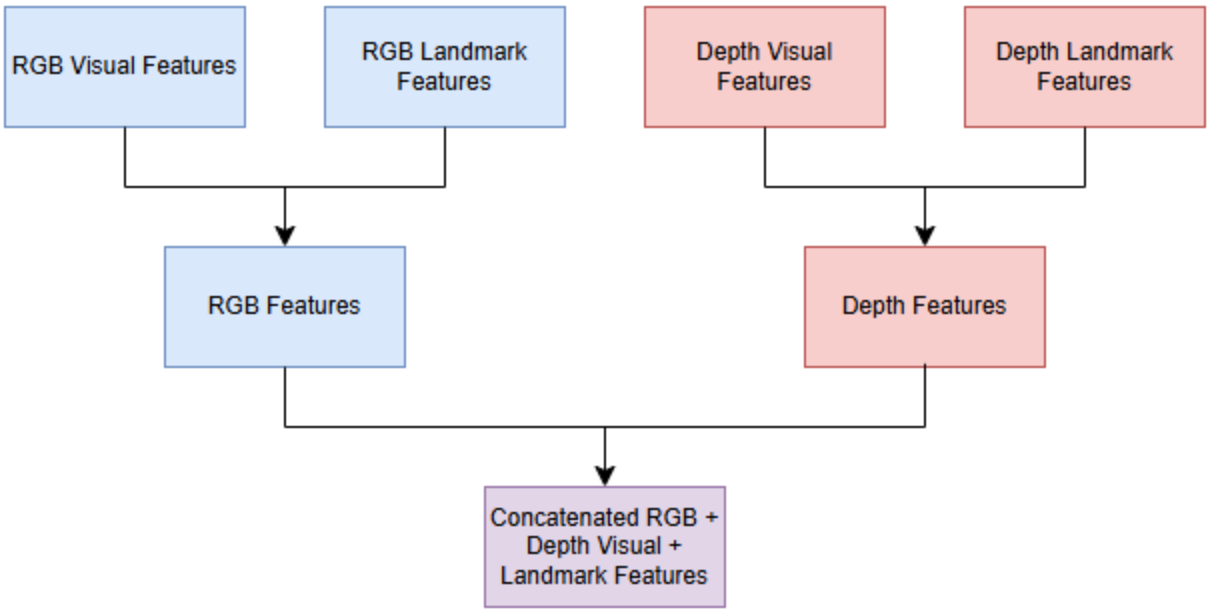

Figure 3.8 Multi-modal concatenation hierarchy.

The features extracted from the InceptionV3 model are the inputs for the subsequent LSTM-RNN classifier. The temporal dynamics of the hand gestures were maintained by arranging the feature vectors in sequence to preserve the order of the frames. Once the feature vectors have been created and saved, they are fed to three LSTM blocks, one for landmarks, one for RGB, and one for depth. This is how the preprocessing pipeline connects to the training pipeline, as seen in Figure 3.9. In training, the LSTMs outputs are concatenated and fed to the final LSTM. The key to connecting the preprocessing to the training is the NumPy file which is a standard binary format used by NumPy to store arrays.

There are two common approaches for feature extraction in vision-based dynamic hand gesture recognition systems. The first is batch or offline feature extraction, and the second is end-to-end or online feature extraction (Oliveira et al., 2017). For batch feature extraction, the CNN-based image encoding process is decoupled from the sequence learning task. Each frame in each gesture sequence is passed through the frozen pre-trained CNN individually, usually outside of the main gesture classification model, with its feature vector saved to disk or memory. The resulting feature vectors are then fed into an RNN or transformer-based model. End-to-end feature extraction integrates the CNN directly into the gesture classification model. The full sequence of frames is fed into the CNN using a TimeDistributed wrapper or 3D convolutional architecture (Lu et al.,

2023). The spatial features of each frame are dynamically extracted through forward propagation and then passed to the temporal layers. In this project, batch feature extraction was used which allowed for faster training runtime and simpler error isolation; however, this required large storage for precomputed features and had limited flexibility for augmentation.

3.3.3 Learning Algorithm

There are two popular machine learning methods that are suitable for temporal image classification problems, long short-term memory, and recurrent neural networks (Hax et al., 2024).

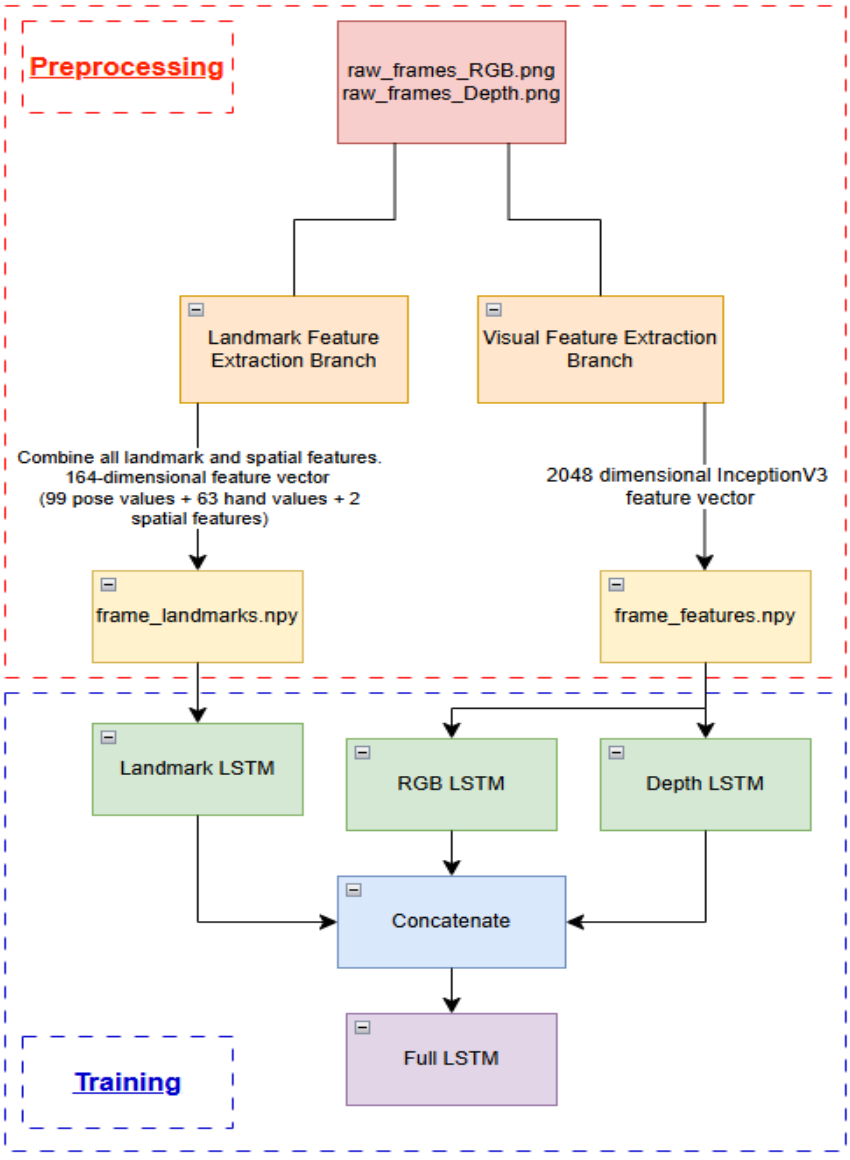

Figure 3.9 Preprocessing to Training Pipeline. All operations shown in the block diagram are described in the text.

Recurrent neural networks are dynamic systems with an internal state at each time step of classification. This is due to circular connections between higher and lower layered neurons in addition to the optional self-feedback connections which all enable RNNs to propagate data from earlier events to current processing steps and build a memory of time series events. The most common method for training RNNs in temporal, supervised learning is backpropagation through time (BPTT). Briefly, for every RNN, there is a feed-forward neural network (FFNN) with identical behavior to it which can be determined by unfolding the RNN in time (Wang et al., 2015). After each training sequence, RNN is unfolded in time, and the error is calculated for the output units and injected backwards into the network with updated weights for each calculated time step. In the RNN, the weights are updated with the sum of its deltas over all time steps.

Standard RNNs are limited to only bridging 5-10 time steps due to the growing and shrinking of the back-propagated error signals every step, with some blowing up and some vanishing. A gradient-based method, Long Short-Term Memory (LSTM) can learn how to bridge minimal time lags with more than 1000 discrete time steps, addressing the vanishing errors.

LSTMs use constant error carousels (CECs) to facilitate a constant error flow within special cells. Suppose there is only one unit u with a single connection to itself (Hochreiter and Schmidhuber, 1997). The w_{uu} represents the weight of the connection from the unit u to itself. The derivative of the identity function contains the *net* term which refers to the total weighted input to the cell. At a single time-step τ with the next time-step $(\tau + 1)$, the local error back flow of u is given by:

$$\vartheta_u(\tau) \;=\; f_u'\big(net_u(\tau)\big)\, w_{uu}\, \vartheta_u(\tau + 1)$$

$$f_u(net_u(\tau)) \;=\; \frac{net_u(\tau)}{w_{uu}}$$

The CEC preserves the internal activation or called state by using the identity function $f_u$and setting a fixed weight $w_{uu} = 1.0$ which may be reset by the forget gate (Wang et al., 2015). In a neural network, the CEC is connected to other units in the network in addition to itself, introducing additional weighted inputs and outputs. As seen in Figure 3.10, the input and output gates scale the input and output respectively. The connections going to the neuron u may have conflicting weight update signals. In this case, LSTM connects the CEC to the network input layer and other memory cells using input and output gates, forming a complex LSTM unit called a memory block.

The input gates have sigmoid threshold units with an activation function range of [0,1], scaling the signals from the network to the memory cell appropriately. When the gate is closed, there is close to zero activation. The output gates control access to the memory cell contents, protecting other memory cells from the disturbances coming from *u.* All gates are controlled by the maintained state, network input, and hidden activation of previous time step.

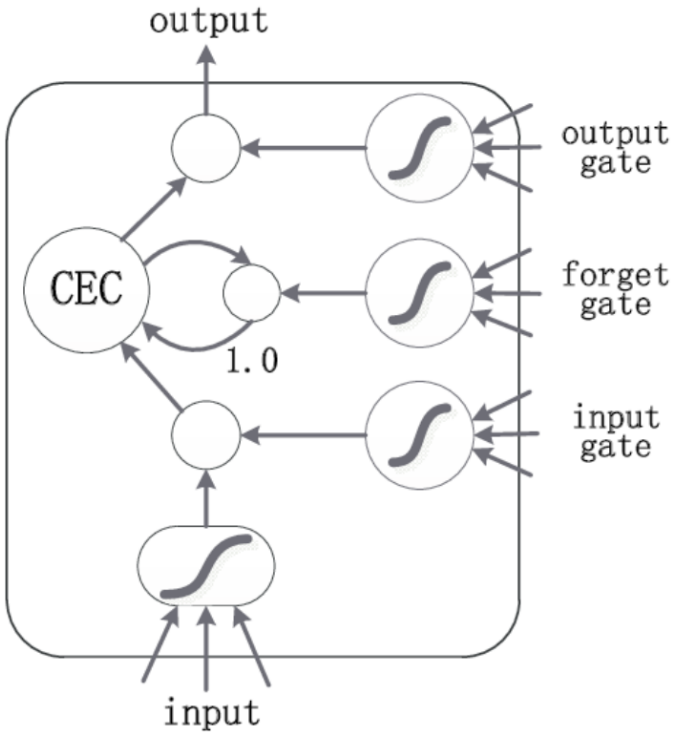

Figure 3.10 LSTM one-cell CEC memory block (Wang et al., 2015).

The Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) is a powerful dynamic classifier.(Staudemeyer and Morris, 2019). LSTM-RNN is able to feed signals from previous timesteps back into the network, making it ideal for tasks with temporal dependency such as recognizing dynamic hand gestures. In the Hax et al. (2024) paper, an LSTM-RNN is used for dynamic hand gesture recognition. In the Yaseen et al. (2024) paper they also take inspiration from Hax's LSTM-RNN, adding MediaPipe before using InceptionV3 while keeping the LSTM at the end of the pipeline.

Sequences of features extracted from the dataset are the input to the LSTM-RNN classifier. Each sequence is comprised of feature vectors corresponding to each frame of the corresponding dynamic hand gesture. The input shape is defined as (sequence_length, feature_dimension), where

sequence_length is the number of frames in each sequence, 40 frames in this case. Meanwhile, feature_dimension is the dimension of the feature vector extracted by the InceptionV3 model.

When looking at the model's architecture layer-by-layer as seen in Figure 3.11, the first one is the input layer which is assigned a predefined shape representing a sequence of feature vectors over time. There are four input branches, RGB visual, RGB landmark, depth visual, and depth landmark. Each of these branches have their own input layer which is passed through a TimeDistributed layer of 256 units and ReLU activation, and then to an LSTM layer with 128 units.

There are multiple LSTMs used in the model architecture. After the four LSTMs, one from each branch, provide their outputs, the visual and landmark branches for each modality are concatenated to two branches, RGB features and depth features. Each of these two branches has its own LSTM layer with 128 units. Finally, the outputs from the RGB features LSTM and the depth features LSTM are concatenated to make the model truly multi-modal. The concatenated features are passed through the final LSTM layer with 256 units to capture any temporal dependencies across the modalities. The fully connected layers at the end are a dense layer with 256 units and ReLU activation, a 50% dropout layer, and a final dense layer with SoftMax activation, allowing classification into six gesture classes.

Dropout layers are used after the LSTM layers in addition to later on after the first dense layer to prevent overfitting. After the LSTM layers, there is a dense layer with 512 units. It introduces a nonlinear transformation to condense the representation before the final output. With a ReLU activation function, this layer applies an L2 regularizer and a dropout rate of 0.5 to prevent overfitting and improve generalization. The final dense layer is the output layer with units equal to the six gesture classes. It uses a SoftMax activation function to produce a probability distribution over the gesture classes, allowing the model to make a final classification decision.

The LSTM-RNN classifier is compiled with an Adam optimizer with a small learning rate for slowly refining weight updates. The loss function used is categorical cross-entropy, appropriate for multi-class classification tasks with accuracy as the performance metric. The training process also included callbacks such as early stopping, learning rate reduction, and model checkpointing to ensure optimal training and prevent overfitting.

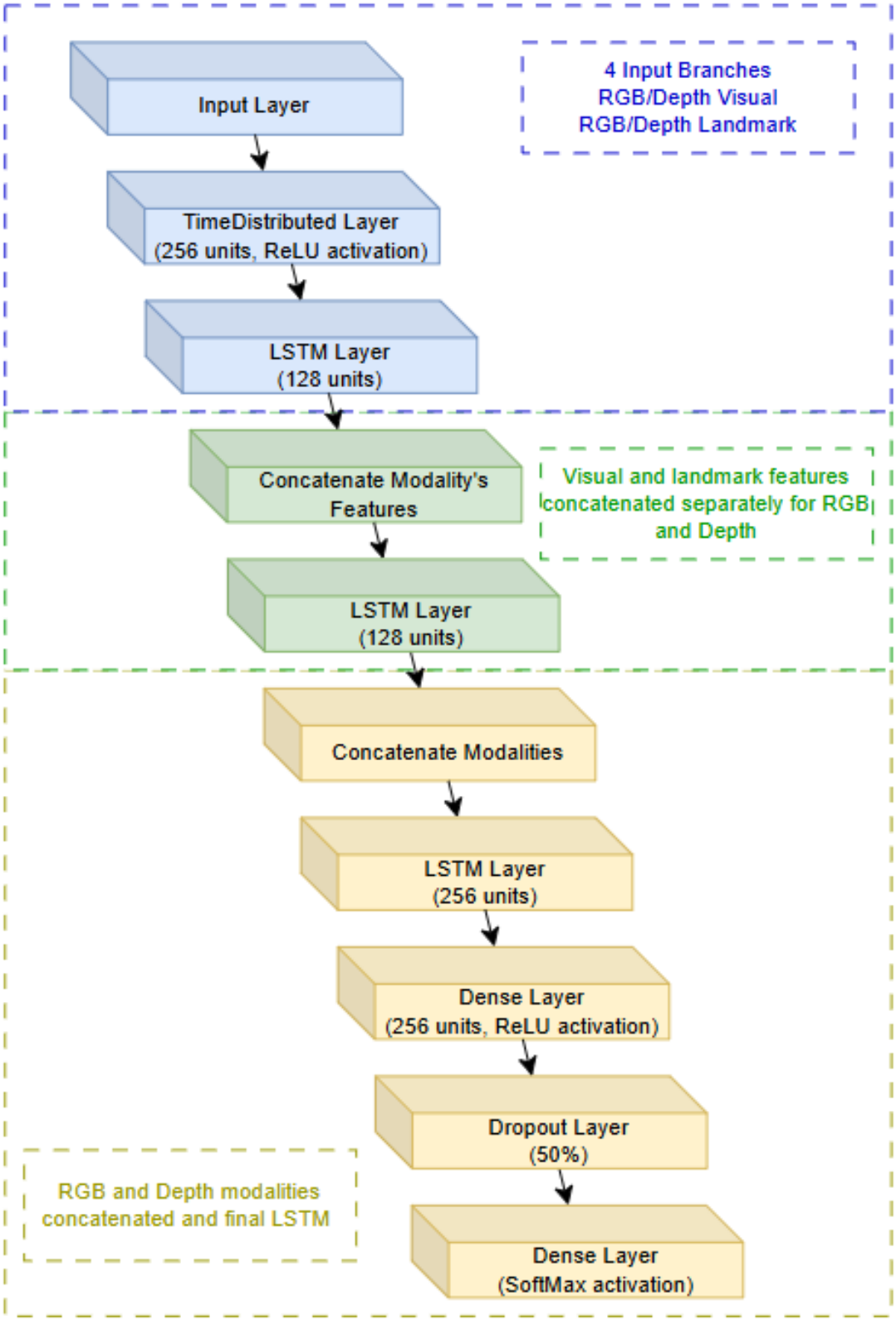

Figure 3.11 Learning algorithm model architecture.

3.3.4 Training and Validation

3.3.4.1 Training

Several key hyperparameters and optimization techniques were utilized to optimize the training process. The hyperparameters in focus are Learning Rate, Batch Size, Epochs, Regularization, and Dropout.

- The learning rate of the Adam optimizer used for this mode was 5 x 10e-5. This is relatively low to help with fine-tuning the model weights, enabling convergence while avoiding large oscillations. The Adam optimizer was chosen due to its ability to adjust the learning rate during training. This optimizer combines the advantages of the AdaGrad and RMSProp algorithms.
- A batch size of eight was used to train the model. This size was selected due to its efficient GPU utilization while fitting within the memory constraints of the Graham cluster.
- The model was allocated a maximum of 100 epochs to train. Training for many epochs allows the model to learn from the data iteratively.
- To penalize large weights and help prevent overfitting, L2 regularization was applied to the dense and LSTM layers with a regularization factor of 0.001.
- Dropout layer with rate of 0.5 reduces overfitting by randomly dropping units during training, pushing the model to generalize better.

The callback techniques that act on validation but are used in training are the following:

- An early stopping callback was used to monitor the validation loss. If the validation loss did not improve for seven consecutive epochs, it is stopped to prevent overfitting and save computational resources.
- Another callback, ReduceLROnPlateau, was used to adjust the learning rate dynamically. If the validation loss plateaued for five epochs, the learning rate would be reduced by a factor of 0.5. This helps the model converge to a better minimum, with better fine-tuning of the model near the end of the training.
- The final callback, ModelCheckpoint, was used to save the model's weights at the epoch with the lowest validation loss. This ensured that only the best performing model was retained for final evaluation.

The hyperparameters and optimization techniques greatly contributed to the model's ability to learn efficiently and effectively from the training data provided while minimizing overfitting and maximizing generalization. The hyperparameter grid included the values seen in Table 3.2. They are typical values that are cycled through the GridSearchCV to determine the optimal combination.

Table 3.2 Hyperparameter Grid Values

Hyperparameter	First Value	Second Value	Third Value
Learning Rate	1e-4	5e-5	1e-5
Dropout Rate	0.1	0.3	0.5
LSTM Units	128	256	x
Regularization Factor	0.001	0.01	0.1

Mixed precision is another technique used for both training and validation. It utilizes both 16-bit and 32-bit floating-point types to reduce memory usage and increase efficiency. The setting used was tf.keras.mixed_precision.set_global_policy('mixed_float16').

3.3.4.2 Validation

Given the sequential nature of dynamic hand gestures, a robust validation approach was essential to accurately assess the model's performance. The sequence-based validation approach uses entire sequences of frames to preserve the complete temporal dynamics and associated movements of each hand gesture iteration.

A single train/test split could lead to misleading results if the train or test set happen to have a lopsided distribution of classes. Therefore, to enhance the validation, stratified k-fold cross-validation was used. Dividing the training dataset into K folds, ensuring that each fold contained a sample representing each gesture. The model was trained and validated five times with each fold acting as the validation fold once and as training for the other four. This approach provides the opportunity for each sequence of the training dataset to be used not only for training but also for validation.

Averaging the performance of all the k-folds leads to a more reliable generalization in the trained model. With the stratified k-fold cross-validation, each fold is guaranteed to contain exactly the same number of examples from each gesture class proportional to the original dataset. This

prevents random splitting that could produce folds with inconsistent proportions of each gesture class, ensuring that no subset is overfitted. Predicted class labels in validation are obtained from the argmax of the probabilities predicted for each class. After all folds are trained, the best model based on highest validation accuracy across all folds is identified and saved along with its training history which contains the loss and accuracy over 100 epochs.

The performance metrics from each fold were aggregated to form an overall evaluation of the model including the mean and standard deviation of accuracy, precision, recall, F1 score, and ROC-AUC.

There are several benefits to using the stratified k-fold cross-validation approach.

- By validating the model on various folds of the data, a thorough evaluation of the model's performance is provided ensuring that no subset was being overfitted.
- Using the entire sequences for validation preserved the temporal relationships between the forty frames allowing the model to learn to recognize the continuous motion of the dynamic hand gestures.

3.4 Testing Procedure

For evaluating the three models, the RGB model, the RGB model with MediaPipe ROI cropping, and the multi-modal model, an evaluation script was used for each model to compare their performances. The test dataset was created consisting of the same structure as the original complete dataset, with 162 instances from each gesture chosen at random. All six gestures, scroll_up, scroll_down, scroll_left, scroll_right, zoom_in, and zoom_out, are represented in the test dataset. Each instance has a 40-frame sequence of the respective gesture recorded.

These scripts loaded each model and the previously separated and unseen test set. As mentioned in the previous section, the dataset was manually split into training/validation and testing instead of the traditional train/test split.

Predictions for the whole dataset are generated by passing the input data through each model. The output probabilities for each gesture class are computed for every sample in the dataset with the result being a 2D array with each row and column corresponding to a sample and the predicted probability for a specific gesture class. The predictions are stored in a dictionary to be used later

in computing the evaluation metrics and generating visualizations. There is no redundant computation as the prediction is reused several times in the evaluation process.

The evaluation metrics are computed for each fold of the dataset during cross-validation using an evaluation function. The inputs are the integer-encoded true class labels from the test set, and the predicted probabilities for the test set are from the output of the models. The three outputs of the evaluation function are the accuracy of the model on the test set, the macro-averaged F1 score across all gesture classes, and the macro-averaged ROC-AUC score across all gesture classes. The gesture class predictions are achieved by selecting the class with the highest predicted probability. The macro-average considers all classes equally, regardless of their relative proportions to each other. The One-vs-Rest (OvR) strategy is used to compute the ROC-AUC scores. Traditionally, the ROC-AUC is designed for binary classification problems; however using the OvR strategy, the ROC-AUC can be made suitable for multi-class problems (Pedregosa et al., 2011). Each classification problem is split into multiple binary classification problems where each class is treated as the "positive" class while the rest are treated as the "negative" class. The ROC curve plots the rate of true positives against false positives, and the AUC curve shows the probability that a randomly chosen positive class receives a higher score from the classifier than a randomly chosen negative class, providing a value of 1 for perfect classification and 0.5 for random guessing, regardless of number of classes.

The confusion matrix is crucial for evaluating the performance of the classification model, by providing a simple view of how the model is performing for different classes, comparing actual labels versus predicted labels, gestures in this case, and unearthing strong and weak points. Through observing the confusion matrix, the gestures with similarities that confuse the model can be pinpointed easily. There are four cases observed in the confusion matrix, True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN), where the true cases represent the model correctly predicting the positive or negative instances, and the false cases representing the model incorrectly predicting positive or negative instances. The definitions of the testing metrics used are shown in Table 3.3.

Table 3.3 Definitions of testing metrics

Metric	Definition	Equation
Accuracy	Measures the proportion of correctly predicted gestures to the total number of gestures.	$\frac{TP + TN}{TP + TN + FP + FN}$
Precision	Measures the accuracy of the positive predictions.	$\frac{TP}{TP + FP}$
Recall	Measures the ability to find all relevant instances.	$\frac{TP}{TP + FN}$
F1 Score	Measures the harmonic mean of precision and recall.	$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$
ROC-AUC	The Receiver Operating Characteristic – Area Under Curve score measures the model's ability to discriminate between classes and provides an aggregate measure of performance across all classification thresholds.	Plots True Positive Rate ($\frac{TP}{TP+FN}$) against False Positive Rate ($\frac{FP}{FP+TN}$)

3.5 Computational Resources

Due to the complexity of the proposed model and the size of the dataset, high-performance computing resources were a necessity. As it is costly and unpractical to purchase a computer with enough power for this job, the next best thing is to obtain the ability to use these powerful computers remotely. The Digital Research Alliance of Canada is a national organization that provides advanced research computing resources and services for the Canadian research community. Available are collections of computer nodes called clusters used to perform computational tasks. They provide CPU and GPUs, and various forms of data storage. The clusters are designed to handle diverse sets of work. Cedar, Narval, and more primarily Graham are all clusters that were used at some point in this research.

The Graham cluster was developed at the University of Waterloo. It provides many CPU resources complemented by GPU nodes equipped with NVIDIA P100 and V100 cards. There is large-scale memory availability and storage space, and it uses the SLURM workload manager for job scheduling. The Cedar cluster was developed at Simon Fraser University. It contains NVIDIA V100 and A100 cards across many compute nodes. The Narval cluster was developed at Calcul Québec in Montreal. It features NVIDIA A100 GPUs with higher interconnect speed and job throughput.

To connect to these clusters, MobaXterm, a terminal and remote desktop application enables interaction with their servers. Over the course of this research, several of the Digital Research Alliance of Canada clusters were taken down for maintenance. However, due to the multiple clusters available, and the Globus file management tool provided by the Digital Research Alliance of Canada, the work was able to be transferred to the Cedar and then Narval clusters and then back to Graham and Nibi.

To submit a training script for a deep learning model, resources must first be specified in a simple Linux utility for resource management batch script (SLURM). Along with the job name, output and error log names, and constraints, the SLURM contains the following resource allocation:

- 1 node.
- 100 GB memory.
- Two NVIDIA V100 GPUs. They are known for their high performance in deep learning tasks.
- 24 hours time.

The training environment required the installation of specific libraries to ensure compatibility and optimal performance. The required modules were standard environment, GCC compiler, CUDA, CUDNN, Python, and OpenCV. A python virtual environment was activated to manage the necessary dependencies and isolate the environment for the training process. Overall, the Graham cluster provided a robust training environment, enabling efficient and effective training for the dynamic hand gesture recognition model. Table 3.4 presents some of the various HPC clusters that were used in this research, providing some information about each.

Table 3.4 The Digital Research Alliance of Canada HPC cluster comparison

Cluster	GPU types	Location	Home Space	High-performance interconnect
Graham	V100-32gb	University of Waterloo, Waterloo, Ontario	16 PB	Mellanox FDR/EDR InfiniBand (56/100 Gbit/s)
Narval	A100-40gb	École de technologie supérieure, Montréal, Québec	35 PB	Mellanox HDR InfiniBand (Up to 200 Gbit/s)
Cedar	V100-32gb	Simon Fraser University, Burnaby, British Columbia	23 PB	Intel OmniPath (100 Gbit/s)
Nibi	H100-80gb, MI300A-128gb	University of Waterloo, Waterloo, Ontario	25 PB	Nokia 200/400G Ethernet (Up to 200 Gbit/s)

3.6 Simulated Robotic Control

This section describes how the precomputed test features were translated into commands for a simulated robotic arm, and how the controller executed each predicted gesture. The gestures were mapped to explicit joint trajectories for the Franka Panda simulated robot in PyBullet.

First, the sequence precomputed feature arrays for the test set which were saved as NumPy files were used to produce a single CSV file. The four streams per sequence, RGB visual, RGB landmark, depth visual and depth landmark features were condensed into the CSV file with each row representing one test sequence. This CSV file is useful for any sort of system that needs the actual gestures and predicted gestures from the model for its application. It centralizes the

predictions, locking down the inputs to the simulated robot to prevent any ambiguity and enabling verification between model outputs and the robot's movements. To create the CSV file, the files in the precomputed test set folder are grouped by normalized instance prefixes, pairing the files for the same instances together. Each sequence is padded to the model's expected length and the same normalization used in training is carried over here. The trained Keras model is run with a one-sample batch for each sequence, saving timing information such as time per frame and time for inference, the sequence index, true gesture and predicted gesture, normalized prefix, the file paths for each of the four NumPy files, the prediction time, and prediction probability. All of this information is in one row with each sequence having its own row. There are 972 rows in the CSV file to represent the 6 gestures with 162 sequences each.

The Franka Emika Panda robot was the chosen simulated robot model to be used as it is open-source and has 7 arm joints plus 2 grippers, allowing for mimicry of the gesture classes in the test set which are performed by human subjects with their arms. Without the two grippers which act as fingers, the full context of the gestures could not be displayed in the simulation. Each gesture is defined as two key joint states, the beginning pose and the end pose. Together these two poses command the robot to set up with a starting pose and an end pose that mimic each of the 6 gestures in the dataset. Linear interpolation in joint spaces between the beginning and end poses over a fixed number of simulation steps and per-step sleep parameters allows the robot to perform the gestures smoothly and accurately. Before performing the gesture, the controller checks that the joint values are within the robot's Unified Robot Description Format (URDF) limits and verifies that the commands have a defined trajectory. If these safety checks fail, the controller logs the sequence and skips the execution.

In the main loop of the robotic control script, the predicted label and predicted probability are extracted, matching the integer label to the gesture name. The script looks for the beginning and end joint vectors for the required gestures and interpolates the beginning to the end, returning to the beginning pose afterwards to keep each gesture independent from the rest. The GUI offers a live rendering of the Franka Emika Panda robot and it can be enabled or disabled. When enabled, it allows for visual verification from the user. When disabled, it allows for faster experimentation.

4. Results

This chapter presents the experimental results of the developed dynamic hand gesture recognition system. There are three models that were used in this work: 1) The model using RGB visualizations only, 2) The model using RGB visualizations and MediaPipe for ROI cropping, and 3) The main multi-modal model proposed in this work.

4.1 Training & Validation Results

From the training and validation of the three models, performance metrics are determined for comparison as seen in Table 4.1 which displays results for the RGB-only model, Table 4.2 which displays results for the RGB model with MediaPipe ROI cropping, and Table 4.3 which displays results for the multi-modal model

Table 4.1 Training & validation metrics (mean and standard deviation across all folds) for model with single input stream RGB visualizations

Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
0.9542 ± 0.0126	0.8393 ± 0.0118	0.5188 ± 0.0105	0.8976 ± 0.0581

Table 4.2 Training & validation metrics (mean and standard deviation across all folds) for the RGB model with MediaPipe ROI cropping

Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
0.9932 ± 0.0009	0.8673 ± 0.0102	0.1469 ± 0.0081	0.6607 ± 0.05

Table 4.3 Training & validation metrics (mean and standard deviation across all folds) for multi-modal model

Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
0.9963 ± 0.002	0.962 ± 0.0102	0.3703 ± 0.0595	0.5129 ± 0.0896

The confusion matrices, shown in Figures 4.1 and 4.2, provide other observable results of the multi-modal model's validation with and without stratified folds, showing that stratification enforces equal splitting amongst folds to prevent class imbalances in any fold of the cross-validation. The confusion matrices are taken from a single fold of the model, chosen at random to

emphasize the effect of stratification in each fold. These plots are not intended to show a comparison between the different models, but just to show the difference that stratification can make when splitting folds with multiple classes.

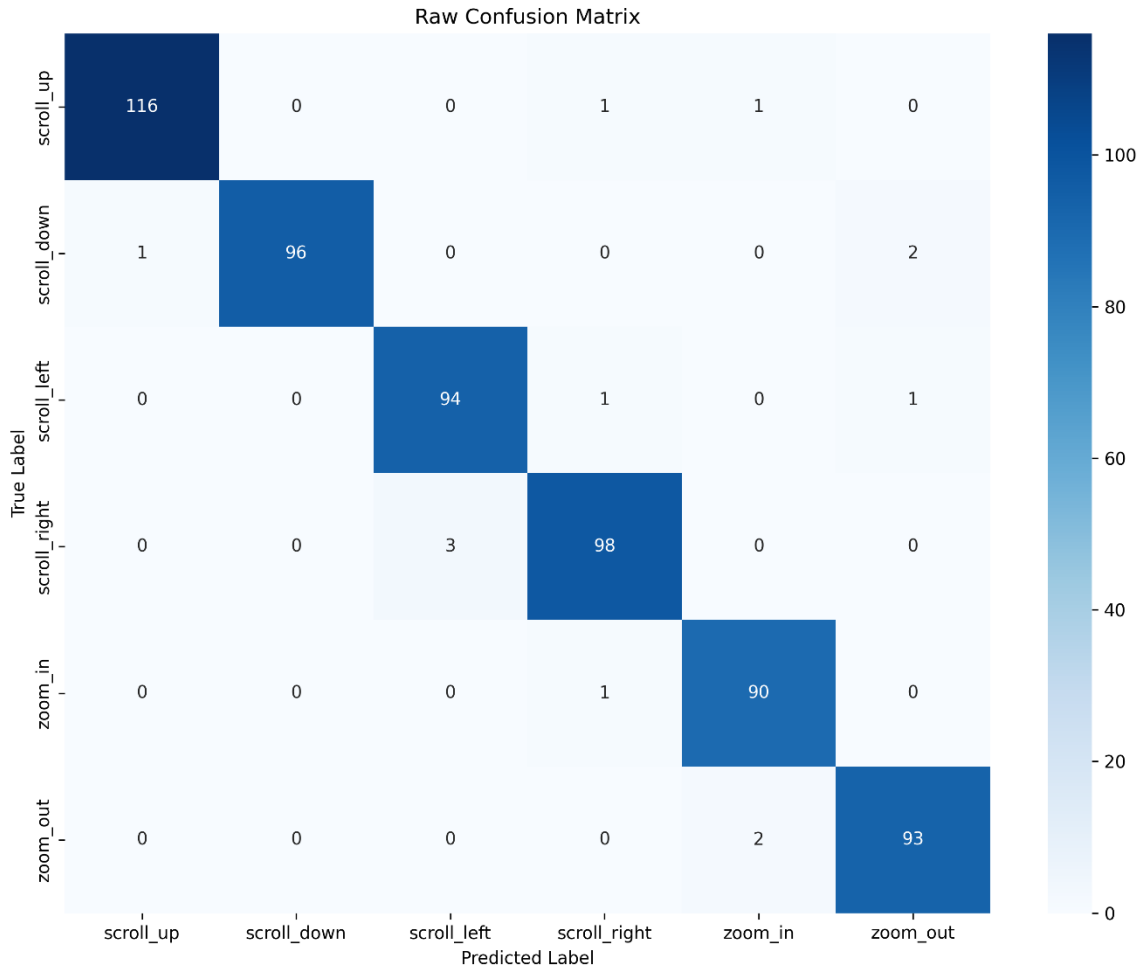

Figure 4.1 Confusion matrix of multi-modal model's validation of a single fold without stratified folds.

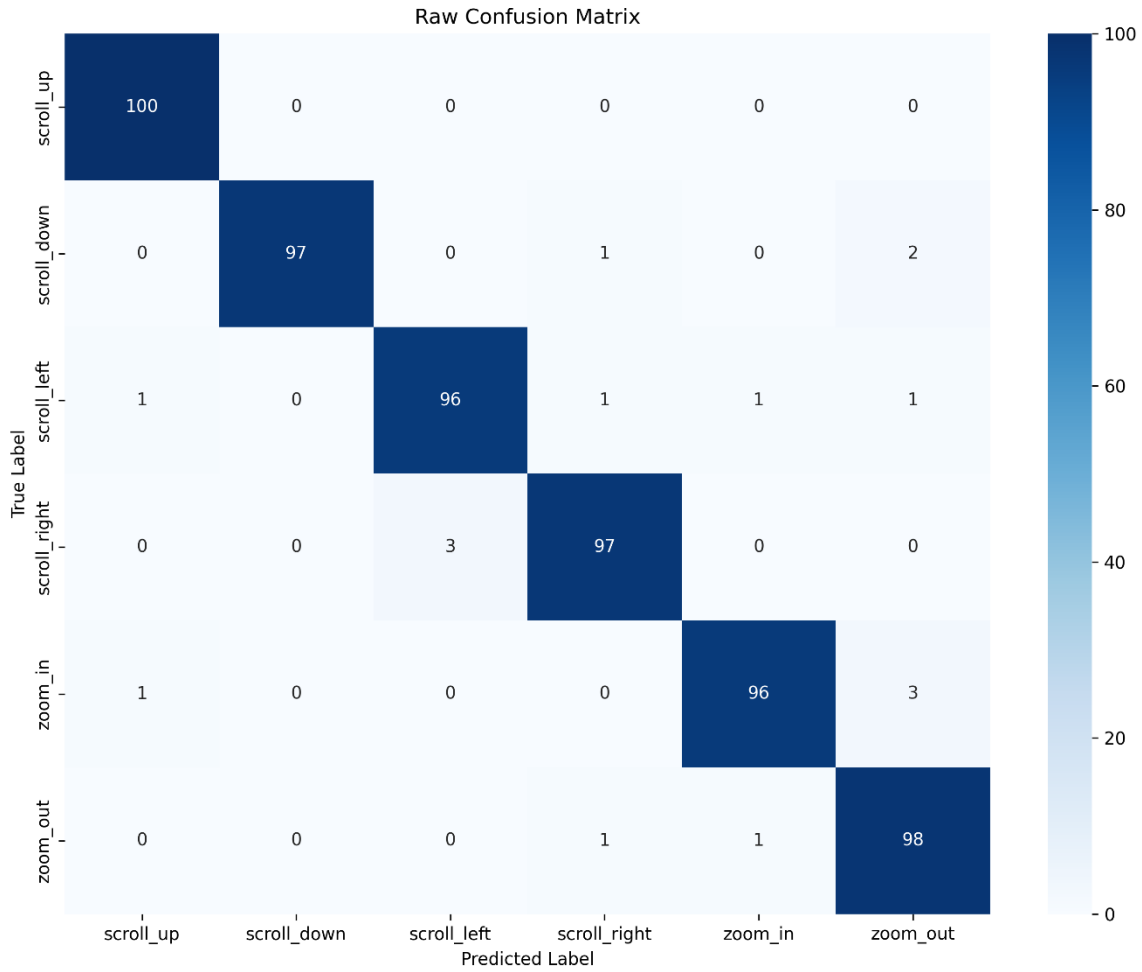

Figure 4.2 Confusion matrix of multi-modal model's validation of a single fold with stratified folds.

The training and validation loss and accuracy plots are determined from the training and validation history of the three models and shown in Figure 4.3 to Figure 4.8. It can be seen that the convergence of the accuracy in the multi-modal model is faster than the RGB-only model but about the same as the model with RGB and MediaPipe ROI cropping. However, the RGB model with MediaPipe ROI cropping has a larger difference between the end values of training versus validation. For the multi-modal model, the loss plot indicates a desired fit where the training loss and validation loss decrease and converge to remain close to each other. The accuracy plot indicates a desired fit where the training accuracy and validation accuracy increase and converge to remain close to each other.

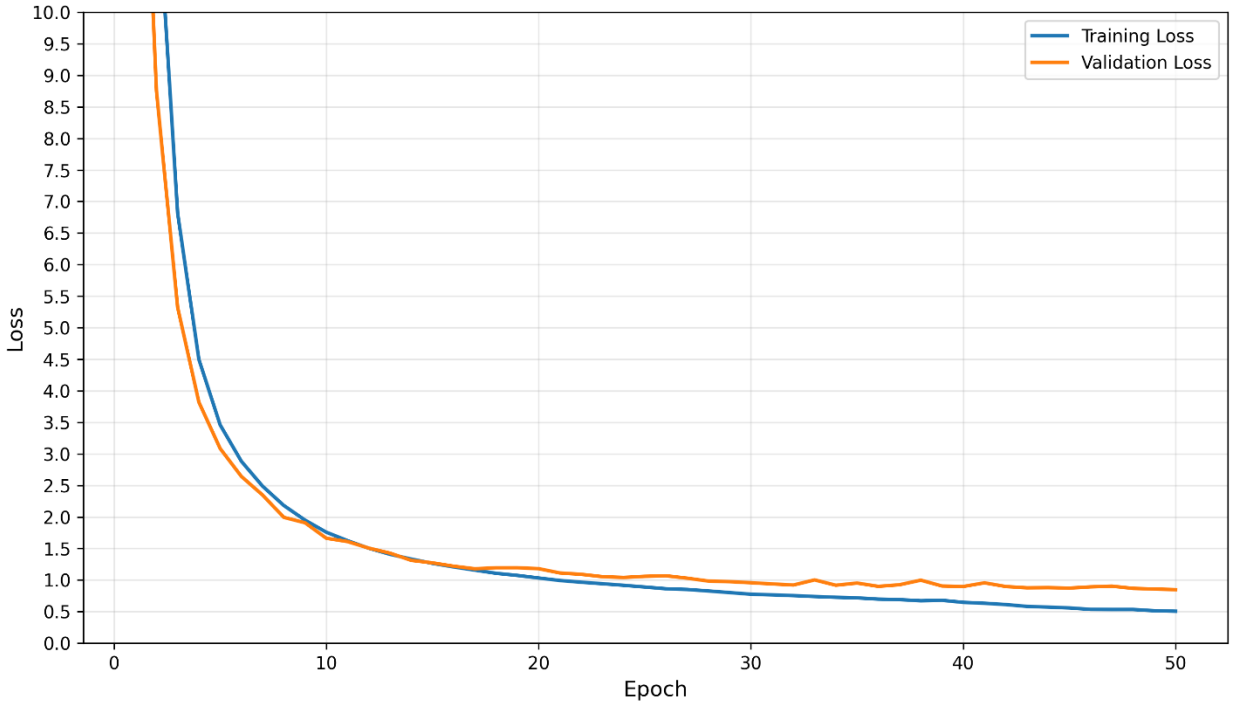

Figure 4.3 Training vs validation loss plot for the RGB-only model.

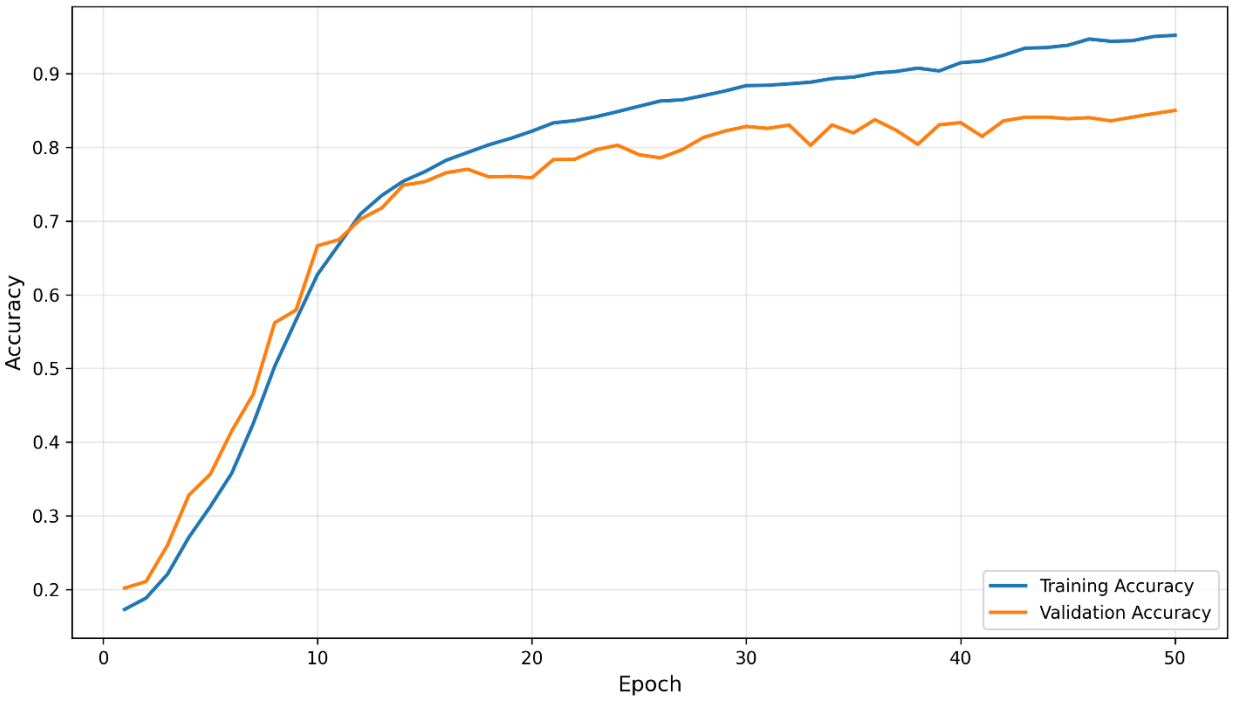

Figure 4.4 Training vs validation accuracy plot for the RGB-only model.

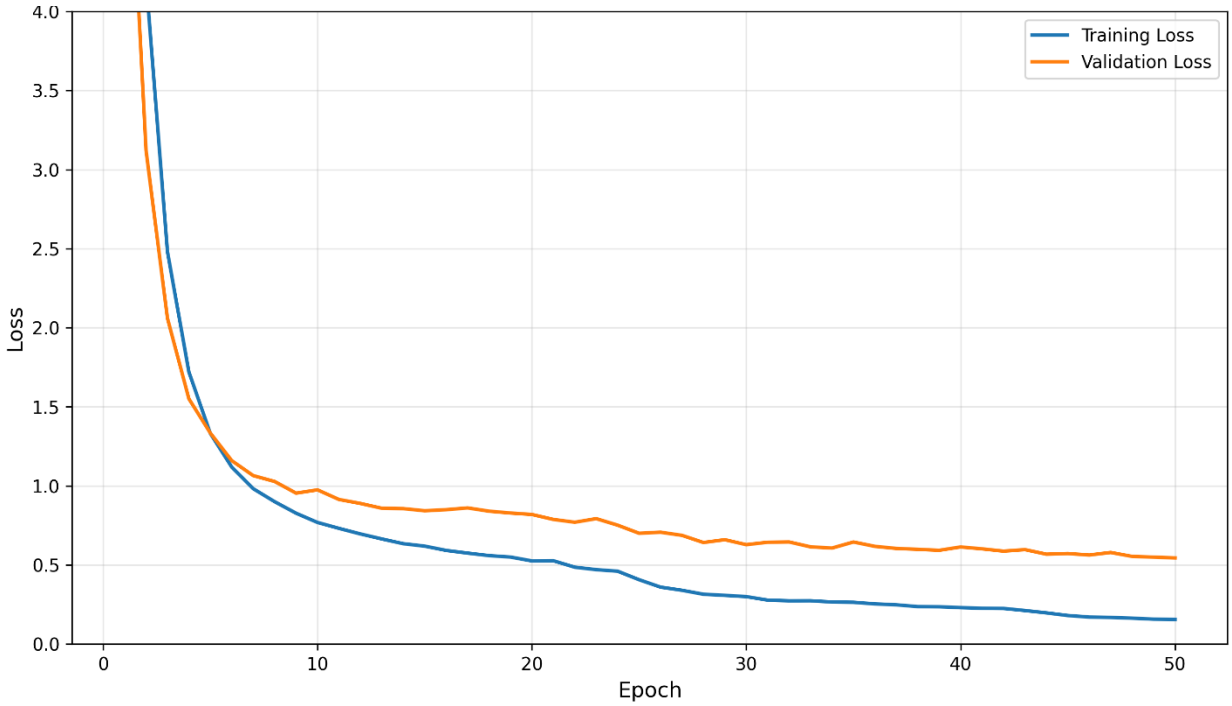

Figure 4.5 Training vs validation loss plot for the RGB model with MediaPipe ROI cropping.

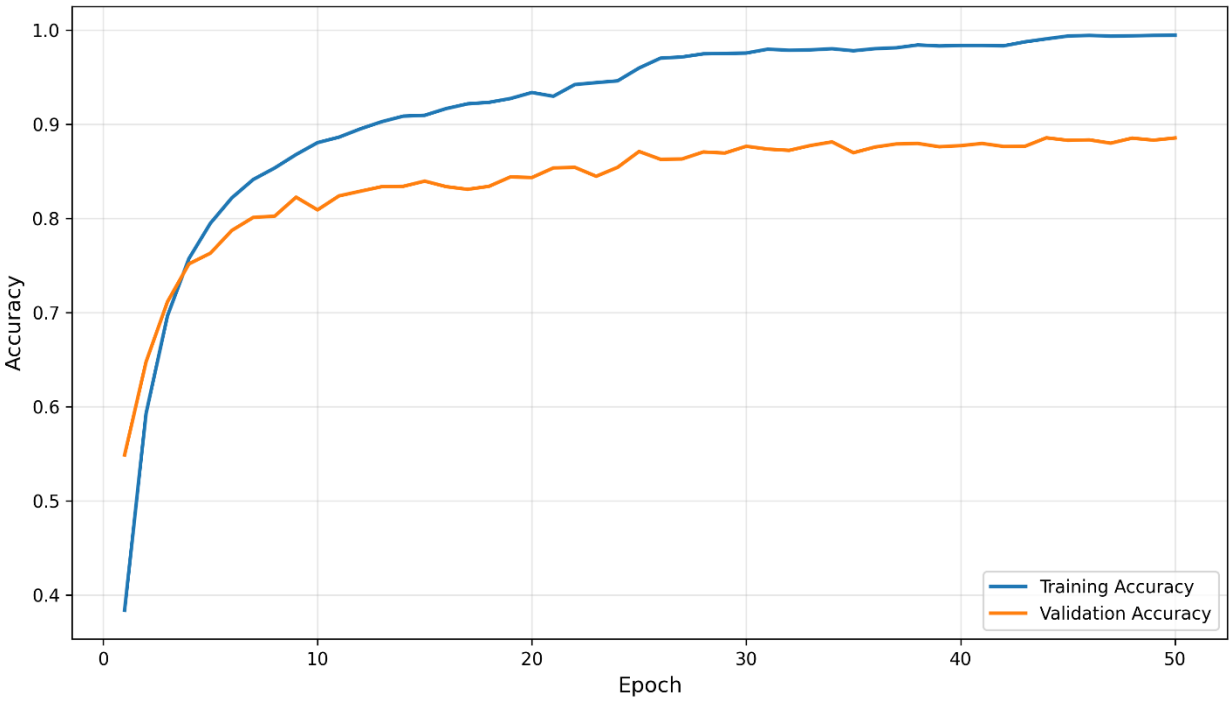

Figure 4.6 Training vs validation accuracy plot for the RGB model with MediaPipe ROI cropping.

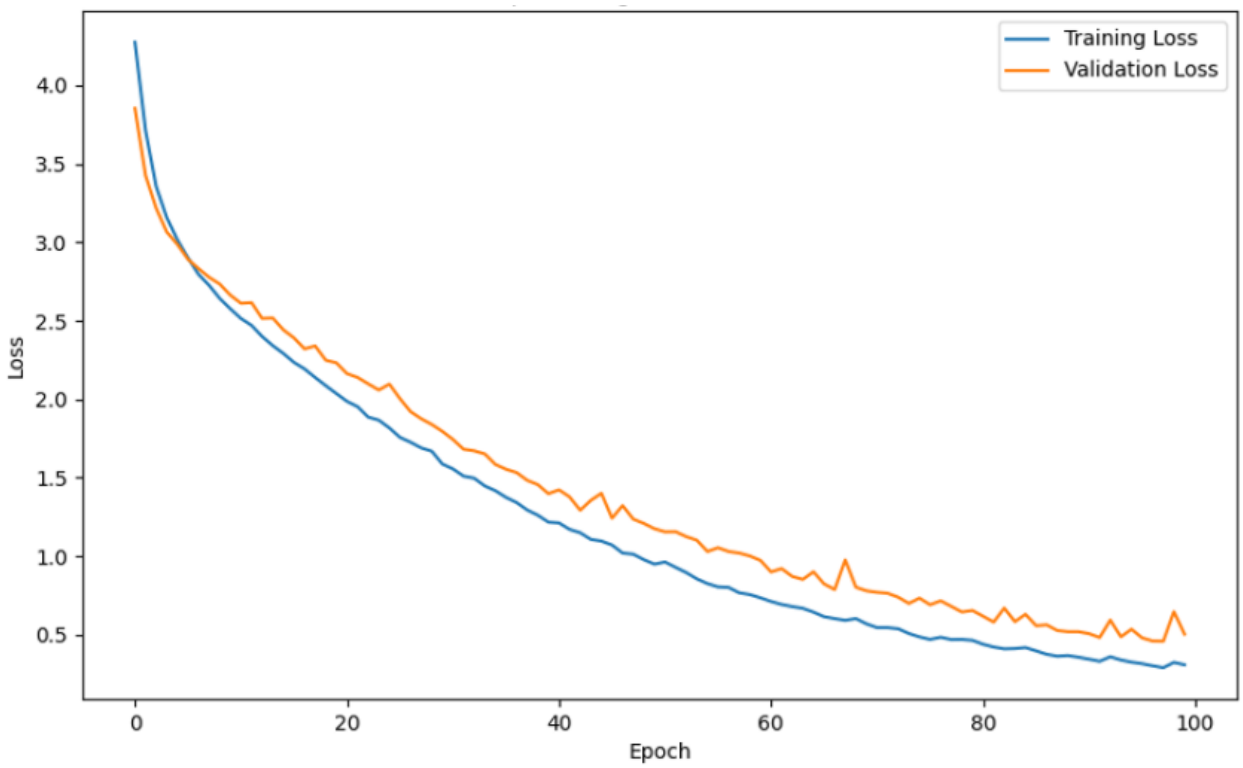

Figure 4.7 Training vs validation loss plot for the multi-modal model.

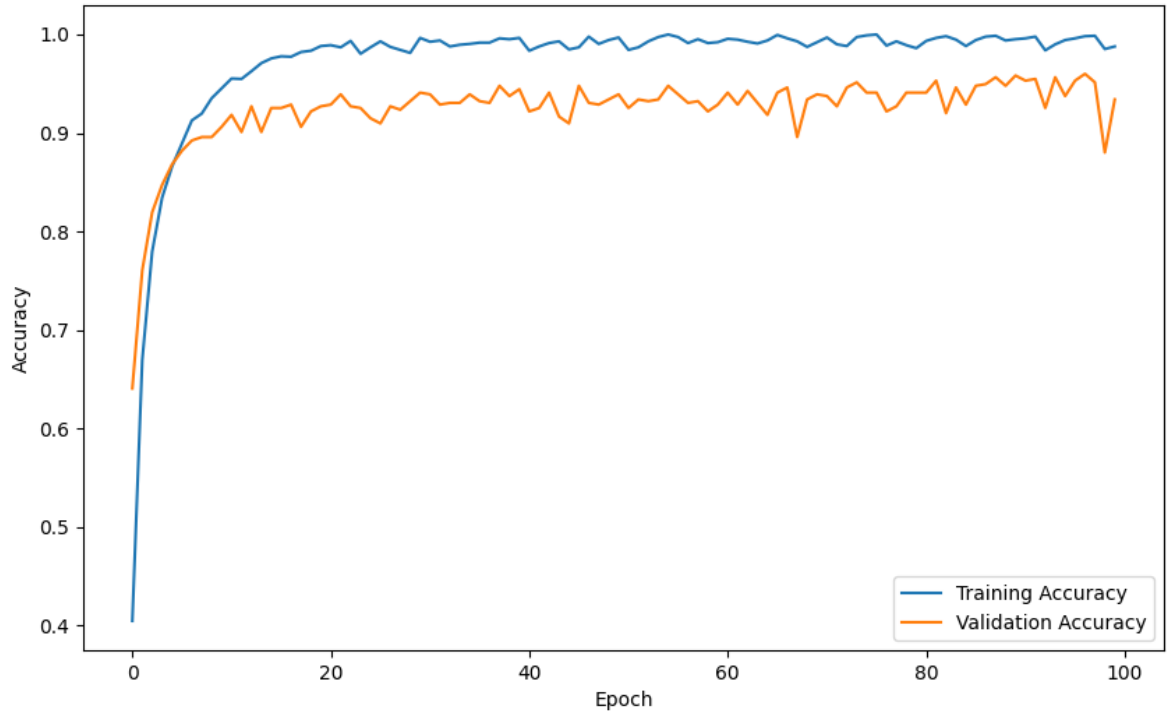

Figure 4.8 Training vs validation accuracy plot for the multi-modal model.

4.2 Testing Results

The test set was separated from the dataset before the model was trained, ensuring that all instances in the test set were unseen by the model before testing.

In the confusion matrix with the true labels as the Y-axis and the predicted labels as the X-axis, the true positives are seen in the diagonal elements where the row and column indexes match. Going along the columns, the number of times that a class was predicted when the true label was something else corresponds to the false positives. Going along the rows, the number of times that a true label was missed due to predicting another class corresponds to the false negatives. For each class, the elements that are not in that class's row or column are correctly predicted as not that class and correspond to the true negatives. The normalized confusion matrix with mean and standard deviation across all 5 folds from testing the RGB-only model is seen in Figure 4.9, from the RGB model with MediaPipe ROI cropping is seen in Figure 4.10, and from the multi-modal is shown in Figure 4.11.

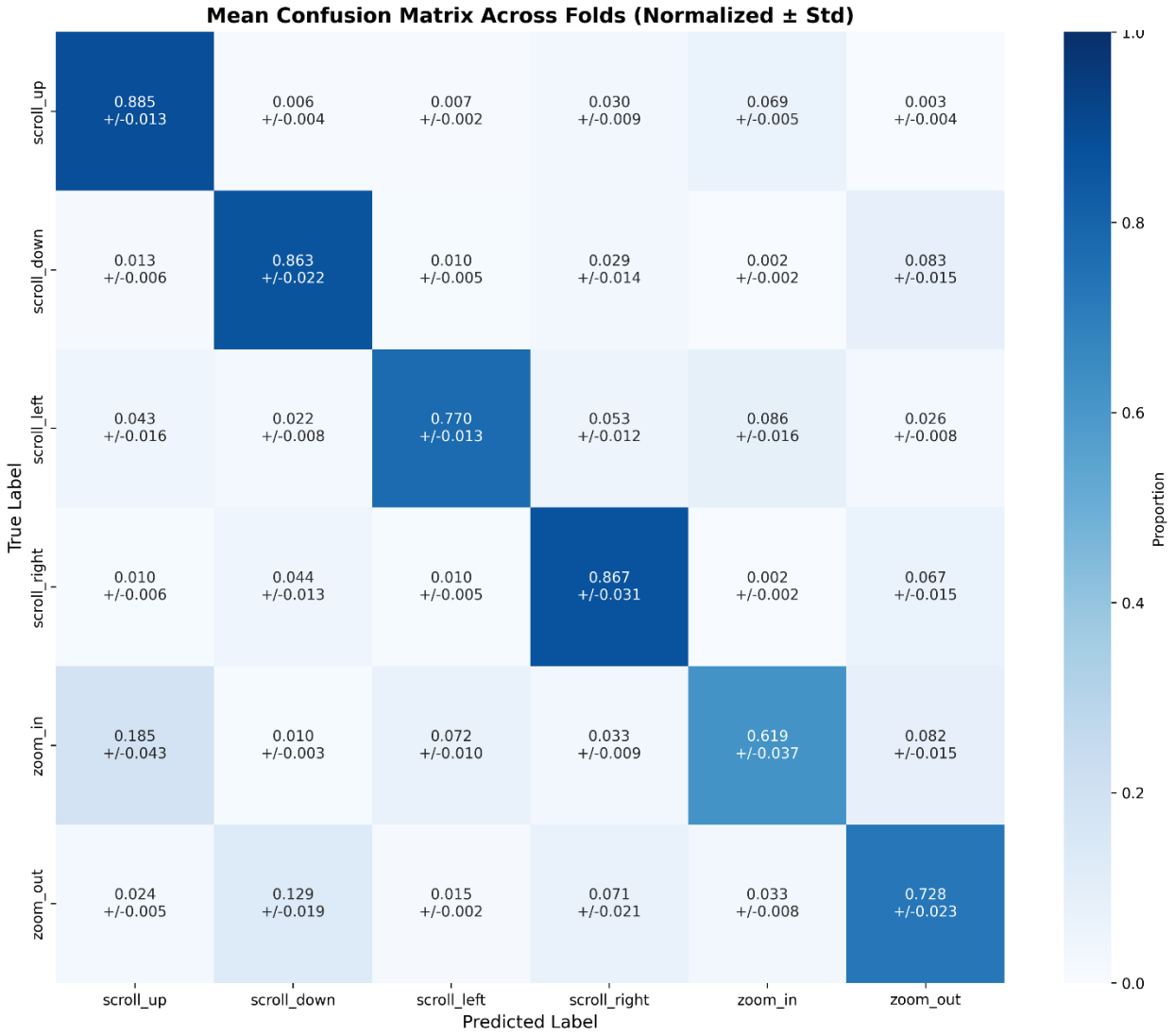

Figure 4.9 Normalized testing confusion matrix with mean and standard deviation across all folds for the single input stream RGB model.

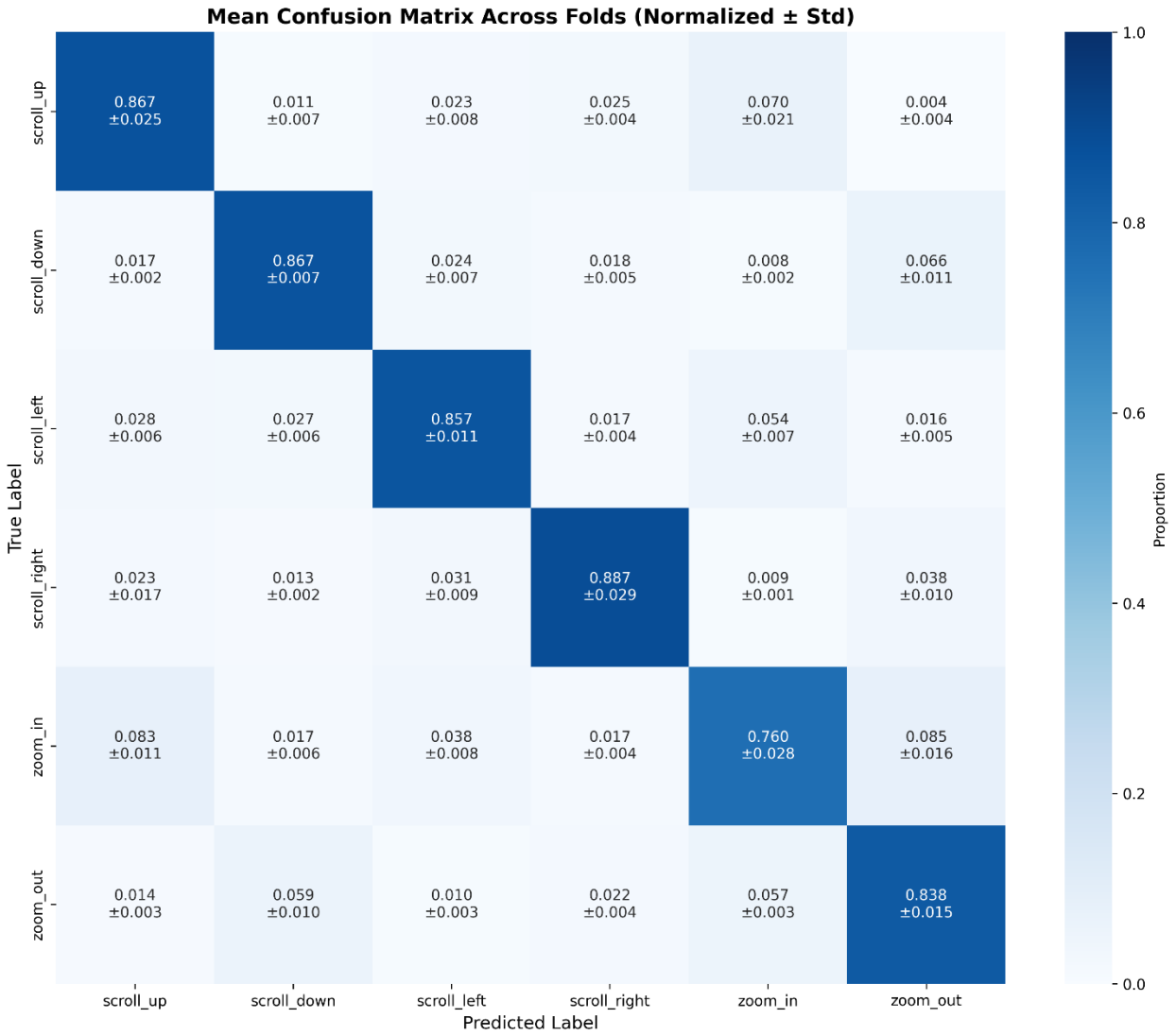

Figure 4.10 Normalized testing confusion matrix with mean and standard deviation across all folds for the RGB model with MediaPipe ROI cropping.

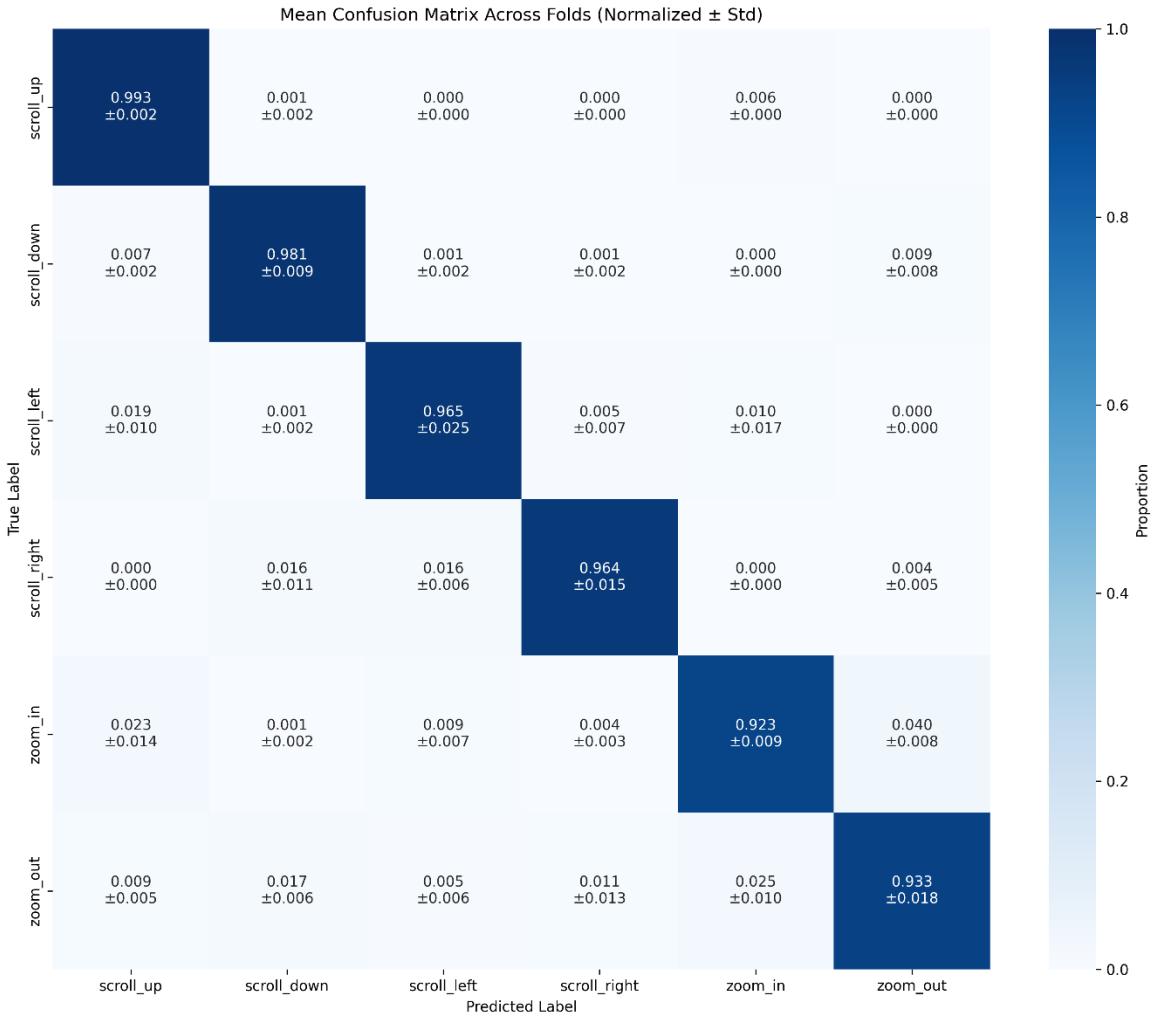

Figure 4.11 Normalized testing confusion matrix with mean and standard deviation across all folds for the multi-modal model.

To provide further details on the performance with the testing data, Table 4.4 shows the performance metrics with mean and standard deviation over all of the folds and gestures for the RGB-only model, Table 4.5 shows them for the RGB model with MediaPipe ROI cropping, and Table 4.6 shows them for the multi-modal model.

Table 4.4 Testing results (mean and standard deviation) over all folds and gestures obtained using the single input stream RGB model.

Accuracy	Precision	Recall	F1 Score	ROC-AUC Score
0.789 ± 0.010	0.791 ± 0.009	0.789 ± 0.010	0.786 ± 0.011	0.961 ± 0.003

Table 4.5 Testing results (mean and standard deviation) over all folds and gestures obtained using the RGB model with MediaPipe ROI cropping.

Accuracy	Precision	Recall	F1 Score	ROC-AUC Score
0.846 ± 0.008	0.846 ± 0.007	0.846 ± 0.008	0.846 ± 0.008	0.977 ± 0.002

Table 4.6 Testing results (mean and standard deviation) over all folds and gestures obtained using the multi-modal model.

Accuracy	Precision	Recall	F1 Score	ROC-AUC Score
0.960 ± 0.007	0.960 ± 0.007	0.960 ± 0.007	0.960 ± 0.007	0.997 ± 0.001

The multi-modal model's performance can also be shown with mean and standard deviation across all folds and gesture classes, as shown in Table 4.7. The testing metrics such as precision, recall, and F1 score are given for each gesture class. The vertical movement gestures, scroll-up and scroll-down have the best recall scores while the horizontal movement gestures, scroll-left and scroll-right have the best precision scores.

Table 4.7 Per-class testing metrics with mean and standard deviation over all folds and gestures obtained using the multi-modal model.

Gesture	F1-Score	Recall	Precision
Scroll_up	0.968 ± 0.008	0.993 ± 0.002	0.945 ± 0.017
Scroll_down	0.973 ± 0.011	0.981 ± 0.009	0.964 ± 0.014
Scroll_left	0.967 ± 0.012	0.965 ± 0.025	0.969 ± 0.013
Scroll_right	0.971 ± 0.008	0.964 ± 0.015	0.979 ± 0.010
Zoom_in	0.940 ± 0.009	0.923 ± 0.009	0.958 ± 0.022
Zoom_out	0.940 ± 0.012	0.933 ± 0.009	0.948 ± 0.014

Figure 4.12, Figure 4.13, and Figure 4.14 show the percentage of true positives, true negatives, false positives, and false negatives for the RGB-only model, RGB model with MediaPipe ROI cropping, and multi-modal model respectively. For all of the gesture classes, there are many more true positives than false positives and false negatives. In all of the gesture classes apart from zoom-

in, there are more false positives than false negatives. A broken Y-axis is used in the plot to ensure that the bar for true negatives does not dwarf the other three bars, keeping them observable.

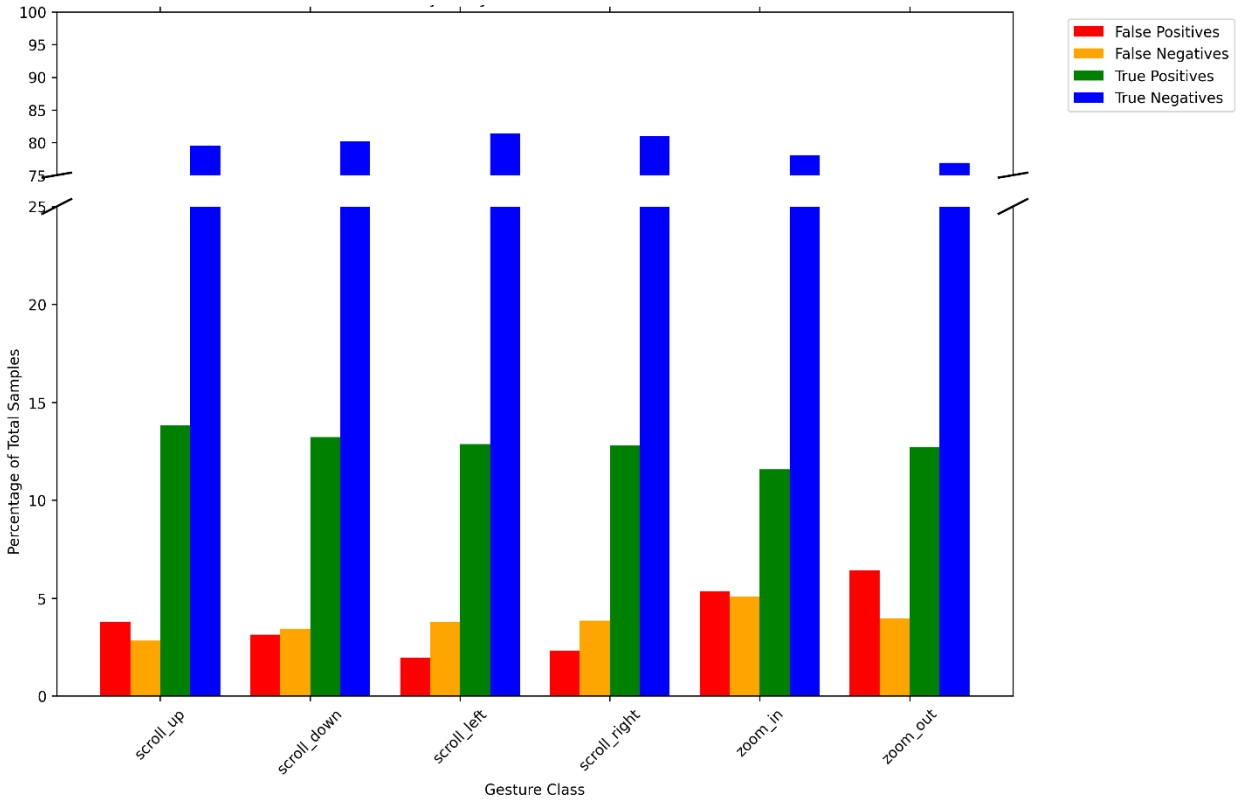

Figure 4.12 Performance Analysis: Percentages of TP, FP, FN, and TN per gesture class for the single stream RGB model.

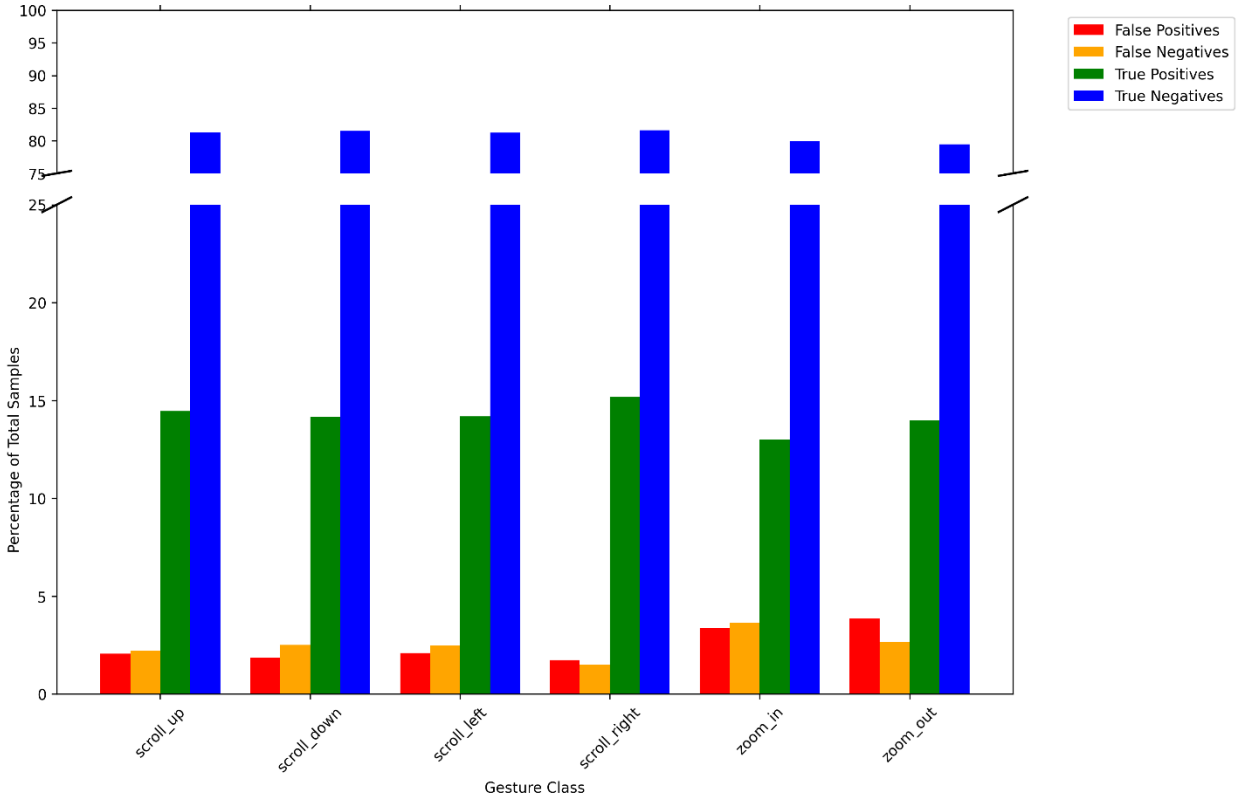

Figure 4.13 Performance Analysis: Percentages of TP, FP, FN, and TN per gesture class for the RGB model with MediaPipe ROI cropping.

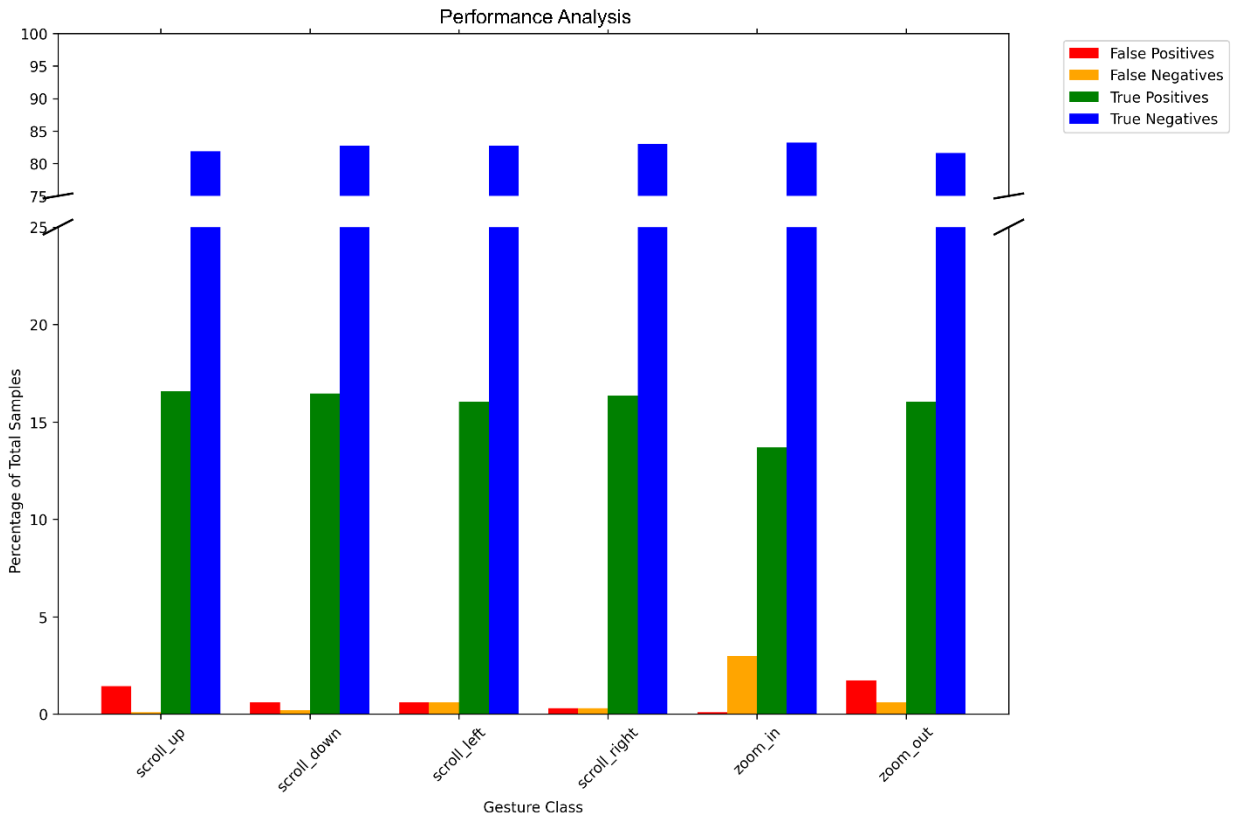

Figure 4.14 Performance Analysis: Percentages of TP, FP, FN, and TN per gesture class for the multi-modal model.

Lastly from the testing results, a two-sided paired t-test was performed to determine the statistical significance of results obtained with the multi-modal model compared to the two single modality models. F1 scores were used as the sample metric because they provide a balanced measure of the performance of each fold across all the gesture classes. For these models, accuracy can also be used since the values are very similar to the F1 scores for each of the folds; however, accuracy does not compare the per-class performance of each fold so F1 scores were the preferred metric for the statistical test. With only 5 folds for each model, the sample size would have only been n = 5. Therefore, each model was run 6 times, each time using a different random seed so that the fold splits would be different for each run while maintaining that model 1's run 1 had the same fold split as model 2's run 1, etc. So with the 6 runs and each run having 5 folds, the sample size was increased to n = 30. Between the multi-modal model and the RGB-only model, the multi-modal model had a significant difference, with a p-value < 0.001, demonstrating that it was significantly better. The same outcome occurred when performing the two-sided paired t-test with the multi-modal model and the RGB-only model with MediaPipe ROI cropping, with ap-value

also < 0.001. These two statistical tests support the statistical significance of the improvement due to the multi-model when compared to the two single modality models.

4.3 Simulated Robotic Control

Figures 4.15 to 4.20 show the simulated robot at the start and end positions for each of the gestures in the dataset. The trajectory of the robot's end effector was plotted for each of the gestures. In Figure 4.21, the trajectories are shown on a uniformly scaled plot that covers a meter of distance to provide a view on how each gesture looked comparatively while being performed by the simulated robot. Figure 4.22 scales the trajectory plots to around 10-15 centimeters with respect to each of the gestures to provide a closer look at the motion of the robot while executing the gesture. Figure 4.23 shows the trajectory plots of the robot's fingers in the zoom in and zoom out gestures. The diverging and converging fingers are a key aspect in differentiating between the two gestures in the dataset and are mimicked in the simulation.

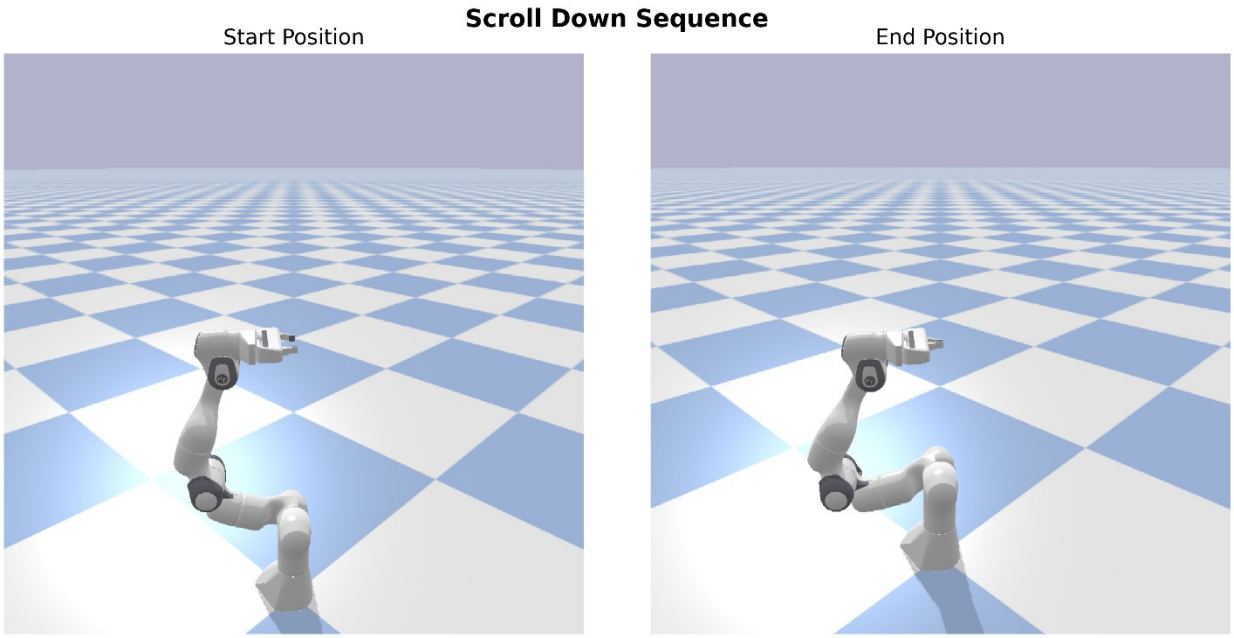

Figure 4.15 Simulated Franka Emika Panda robot assuming the start and end positions for the scroll down gesture.

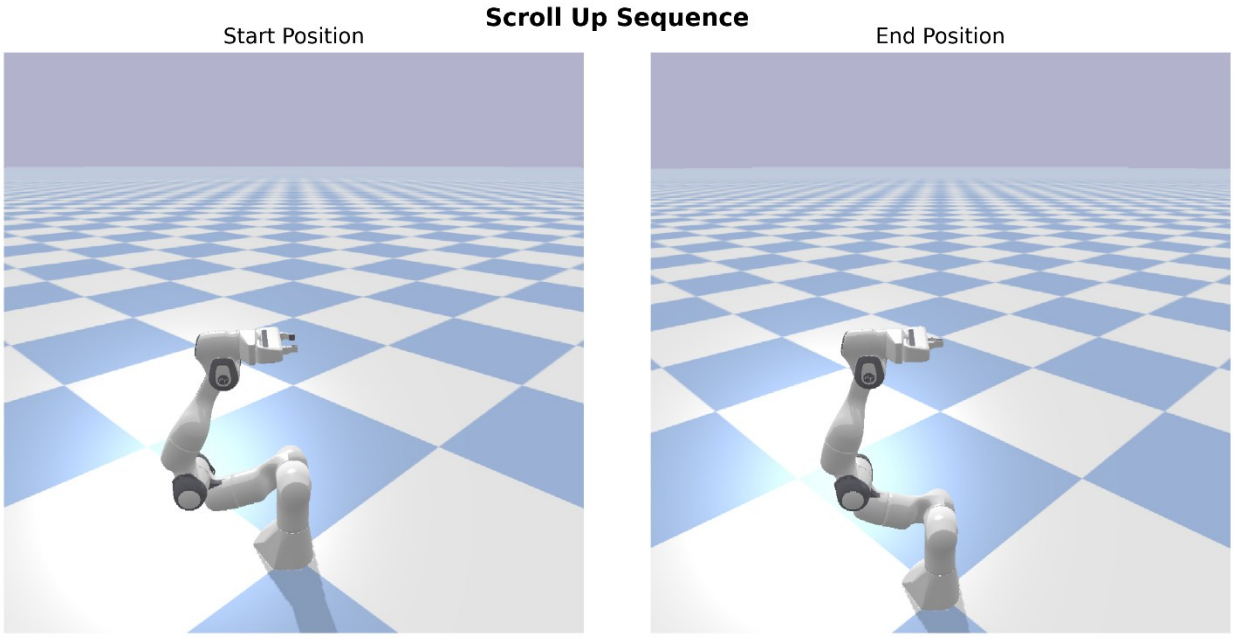

Figure 4.16 Simulated Franka Emika Panda robot assuming the start and end positions for the scroll up gesture.

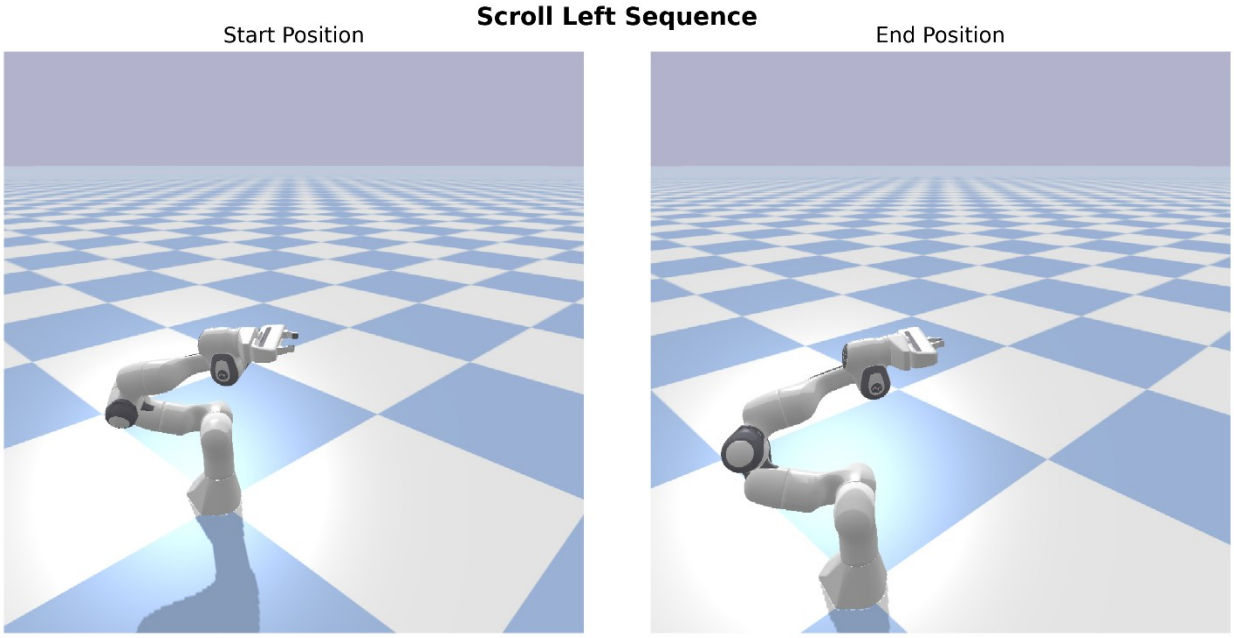

Figure 4.17 Simulated Franka Emika Panda robot assuming the start and end positions for the scroll left gesture.

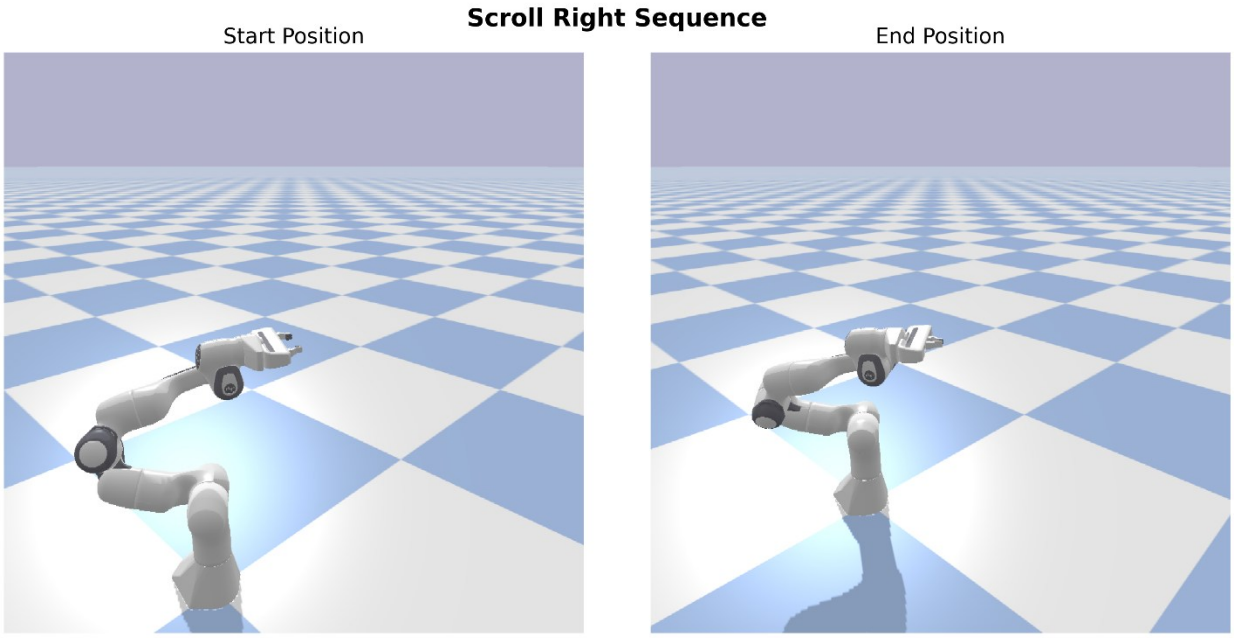

Figure 4.18 Simulated Franka Emika Panda robot assuming the start and end positions for the scroll right gesture.

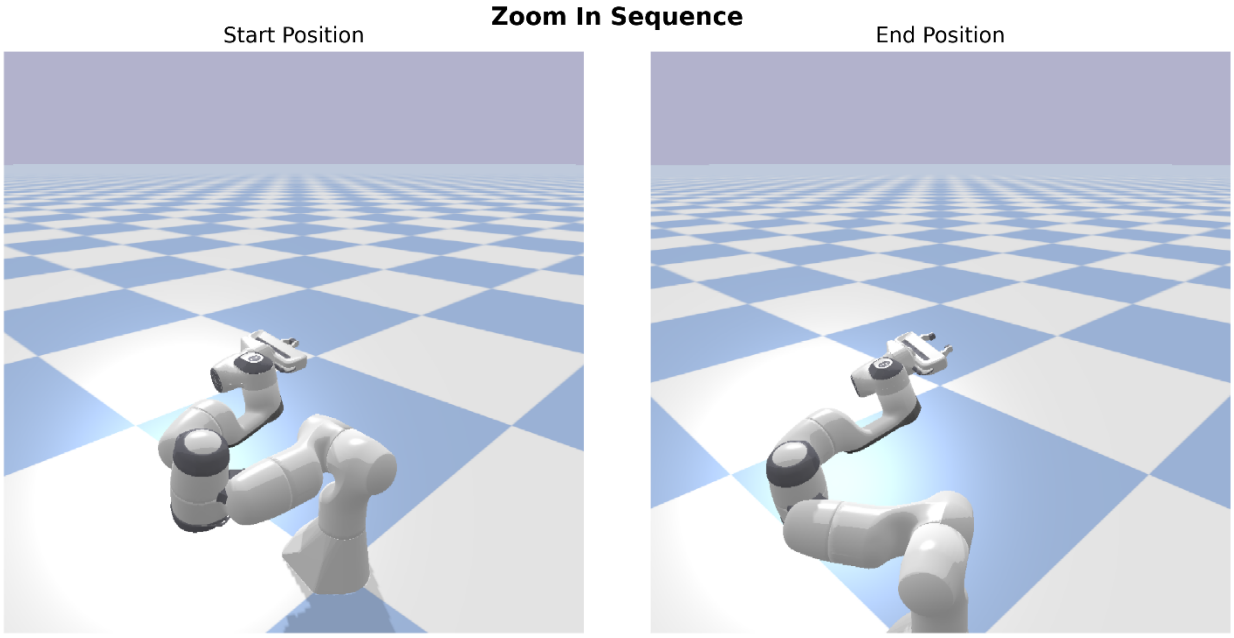

Figure 4.19 Simulated Franka Emika Panda robot assuming the start and end positions for the zoom in gesture.

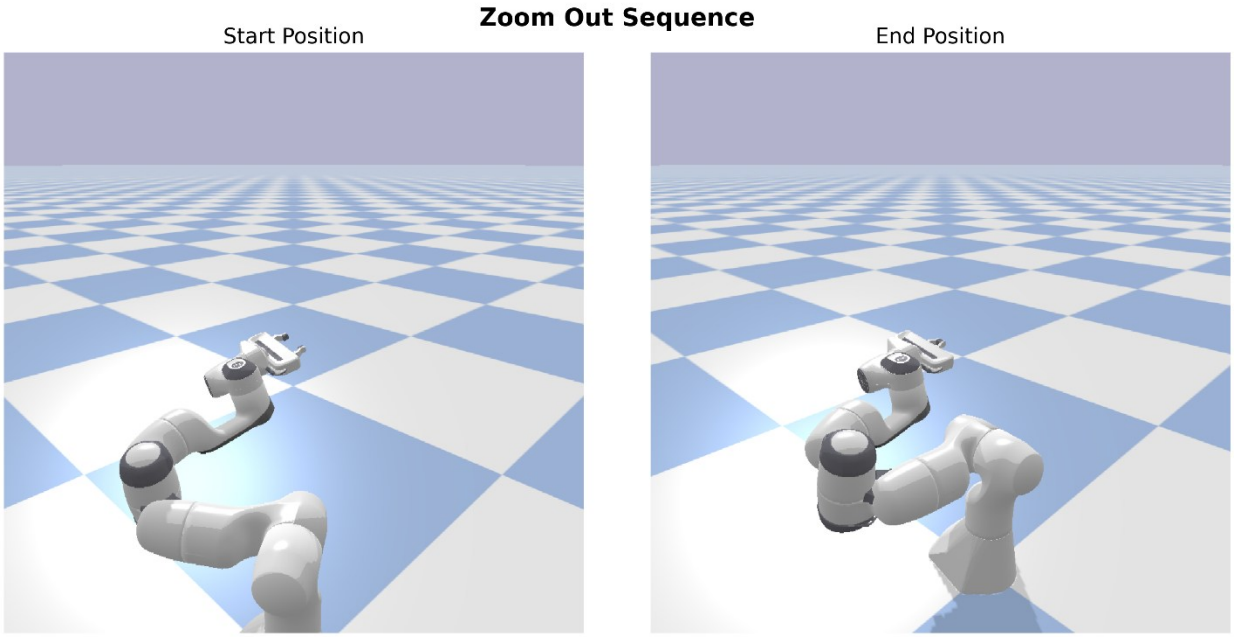

Figure 4.20 Simulated Franka Emika Panda robot assuming the start and end positions for the zoom out gesture.

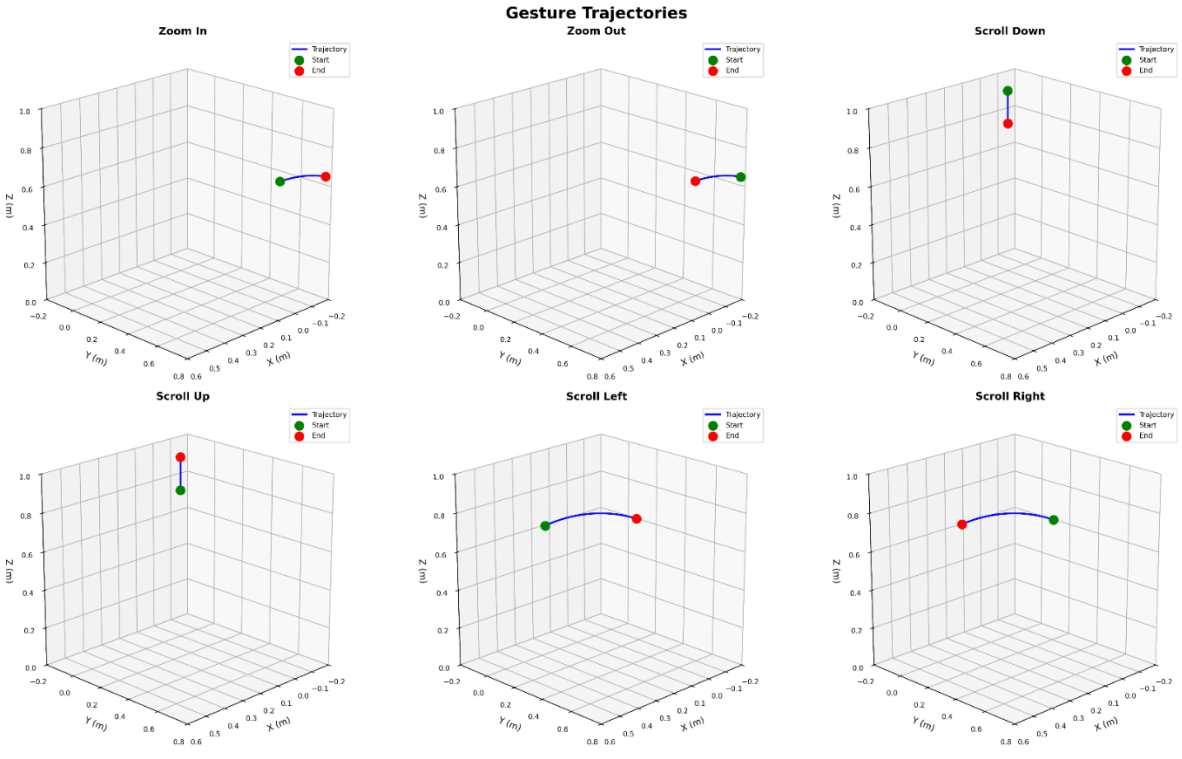

Figure 4.21 Plots of end effector trajectories for each gesture in dataset on plots that are uniformly scaled to a meter.

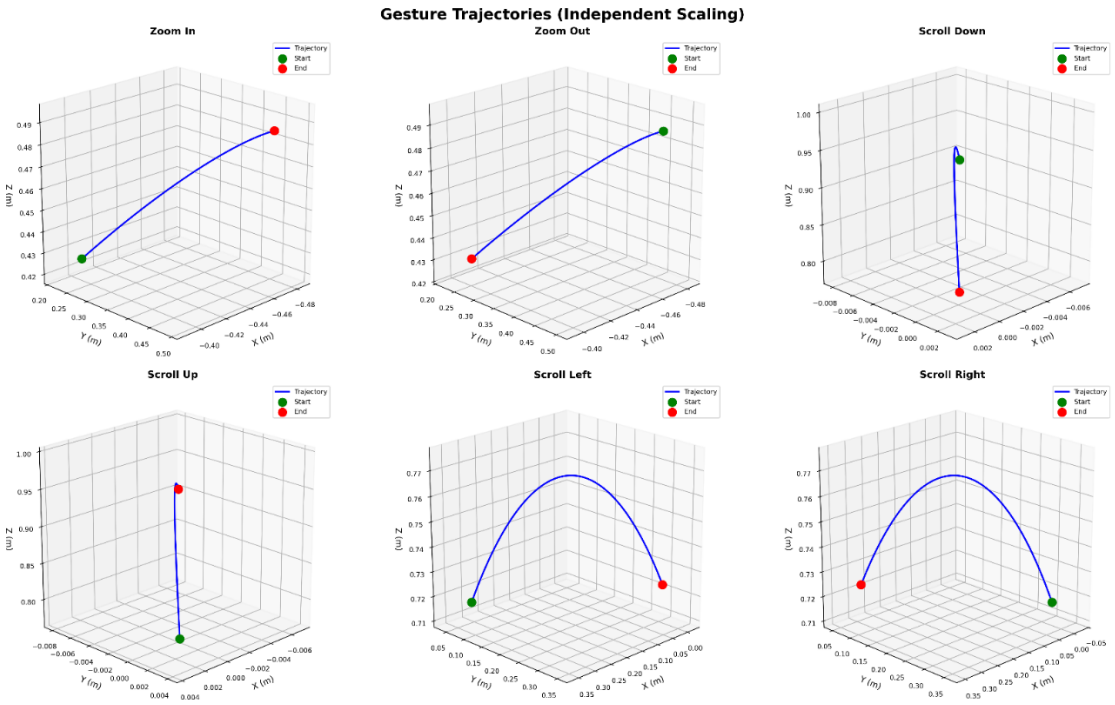

Figure 4.22 Plots of end effector trajectories for each gesture in dataset on plots that are appropriately scaled to within 10-15 centimeters.

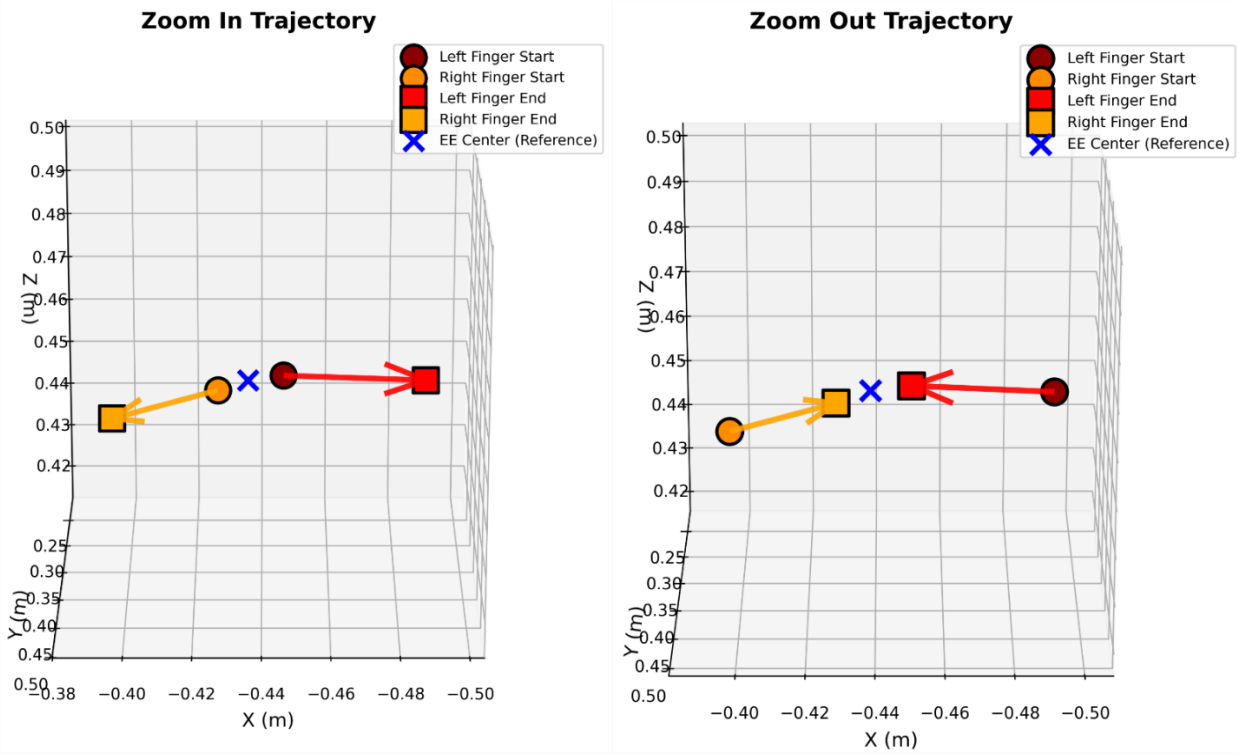

Figure 4.23 Plots of finger trajectories for zoom in and zoom out gestures showing fingers diverging and converging.

5. Discussion

The test set created without any leakage to the training set went through the same feature extraction process as the training set, and all 162 instances in each gesture class for each modality were able to be fed to the model to obtain testing results. Overall, the testing of the model provided successful results, outperforming previous studies.

The performance metrics from testing using the multi-modal model averaged over all gestures are shown in Table 4.6. The obtained accuracy of 96.0% is an improvement over the Hax et al. (2024) paper, where they were only able to achieve 83.7%, and the Yaseen et al. (2024) paper where they achieve 89.7%.

5.1 Multi-Modality & Data Loss

In the Hax et al. (2024) paper, they only use the RGB versions of each sequence in the dataset and only 10/40 frames per sequence. Yaseen et al. (2024) expand on the Hax et al. (2024) work by continuing to use only the 10/40 frames per sequence of the RGB modality from the same dataset while contributing MediaPipe as an ROI extractor to reduce the image dimensions to focus on the hands. In both works, only the RGB images were used from the dataset with the depth images remaining unused in contrast to the multi-modal fusion model used in this work. When handling data for machine learning applications, it is important to keep the split of data between training and testing without them leaking into each other to avoid overfitting, misleading training performance, and poor generalization. In the two studies previously mentioned, the train/test split was done sequence by sequence rather than splitting frame by frame and this practice was continued in this work. With the addition of the depth modality in this work, there was an additional requirement to ensure the two modalities of each sequence were present in either the training or test set but not both. In Hax et al. (2024) and Yaseen et al. (2024), only every fourth frame in each 40-frame sequence was used to train the model to create 10-frame sequences and then from those sequences 70% were used for training, 20% for validation and 10% for testing. In contrast, each whole 40-frame sequence was used to train the model in this work.

Frame-based validation isolates individual frames from the sequence of gestures. However, if temporally adjacent frames from the same gesture sequence end up in both the training and the test set, the training accuracy and generalization may be inflated. In DHGR, frame-based splits give

models a false sense of success by learning to match individual poses rather than the underlying temporal patterns that define the gestures. Sequence-based validation and stratified k-fold cross-validation ensures the model will learn to interpret the temporal dynamics of gestures more effectively by never splitting a gesture video sequence across training and testing. As a result, more honest performance estimates may be obtained along with a model with the ability to learn temporal dynamics.

In the multi-modal model, there are 162 instances for each gesture contained in the test set, with the same instances from the depth modality being taken from the RGB modality to ensure matching instances in the test set, and to ensure that there is no leakage of instances having their depth modality version in the training set while the RGB version is in the test set or vice versa.

Several metrics were used for evaluating the three models in their training, validation, and testing, with the aim of observing any difference between them. The metrics, including accuracy, validation accuracy, loss, validation loss, precision, recall, F1 score, and ROC-AUC score, were compared between the three models to distinguish any large discrepancies in performance. The confusion matrices provided insight into the performance of the models on each class and each fold in 5-fold cross-validation. The results of the training and validation metrics for the model with leakage and without leakage are shown in Tables 4.1, 4.2, and 4.3. When looking at the training and validation metrics for the three models, the training accuracy and validation accuracy of the RGB model are the lowest (95.4% and 83.9%), with the RGB model with MediaPipe ROI cropping being in the middle (99.3% and 86.7%), and the multi-modal model having the highest (99.6% and 96.2%). Additionally, the training and validation losses of the multi-modal model are the closest together (14.3% difference) while the RGB model (37.9% difference) and the RGB model with MediaPipe ROI cropping (51.4% difference) had larger differences. Presumably, a lower difference between training and validation losses would improve the testing metrics and also reduce the overfitting. With the results of testing the three models, it can be observed that this assumption holds true. There are major improvements in every testing metric of the multi-modal model when compared to the RGB model and RGB model with MediaPipe ROI cropping as seen in Tables 4.4 to 4.6. This is because the multi-modal model makes use of the depth modality in the dataset along with the RGB and the MediaPipe hand landmarking, leading to better performance when faced with unseen data.

The confusion matrix further provides insight into the improved performance of the model when trained using both modalities. There are fewer FPs and FNs and more TPs and TNs compared to the RGB model and RGB model with MediaPipe ROI cropping, particularly in the zoom_in and zoom_out gestures. This indicates that the multi-modal model is able to generalize and learn better, especially for the more intricate gestures that may be difficult to distinguish. Gestures such as zoom_in and zoom_out are visually similar which caused the model confusion when only presented with the RGB modality and with every fourth frame of a gesture sequence. Additionally, in Figures 4.1 and 4.2 the difference that the stratification of the folds makes in the cross-validation is shown. Without stratification, there is not a uniform number of gestures in each fold of the cross-validation. When it is stratified, all of the rows will sum up to 100, providing a more accurate validation of the model.

In the testing process, several results were obtained. These included class-wise metrics in a bar graph, confusion matrix heatmap, and performance metric tables. The testing process was aimed at discerning the differences between the testing performances of the RGB model versus the RGB model with MediaPipe ROI cropping versus the multi-modal model, revealing a clear discrepancy between them as the multi-modal model performs much better than the other two.

In Figure 4.11, the confusion matrix from testing the multi-modal model is presented. It can be observed that the scroll_up and scroll_down gestures perform slightly better than the rest of the gestures. However, overall, the model performs well in testing when considering the confusion matrices. It can be observed that there are few FPs and FNs but there are more TPs and TNs. In the RGB model and the RGB model with MediaPipe ROI cropping, the confusion matrices shown in Figure 4.9 and Figure 4.10 indicate that the zoom_in and zoom_out gestures perform worse than the rest reaffirming the observations that the model is more prone to confusion when dealing with these gestures as seen in the Hax et al. (2024) and Yaseen et al. (2024) studies. The performance analysis for the RGB model in Figure 4.12, the RGB model with MediaPipe ROI cropping in Figure 4.13, and the multi-modal model in Figure 4.14 also show that the lowest percentage of FPs and FNs in each gesture class are in the multi-modal model, with the RGB model having the highest percentage and the RGB model with MediaPipe ROI cropping landing in between the two. This observation validates that the multi-modal model can alleviate some of the difficulty that the RGB-only model faces with the zoom_in and zoom_out gesture classes during classification. In

these performance analysis plots, the TNs are not very informative as they just show when the model is correctly predicting that a sequence is not a certain gesture. Due to this, most of the predictions are TNs. It is more informative to focus on the FPs and FNs as they show which gestures or sequences the model is not able to handle well.

5.2 Robotic Control

The executed gestures were visually distinct enough to see due to the distances that the gripper moves between the beginning and end poses, and the correct predictions did not produce any ambiguous or incorrect robot poses due to the explicit joint trajectories designed for each gesture. The trajectories of the robot's gripper for each of the gestures are tracked and plotted as seen in the figures of the previous chapter. The timing and responsiveness of the model fit within the needs of the experiment. The gestures were performed smoothly and consistently with the fixed 240 simulation steps per action, with a mean actuation latency of 1.069 seconds. More simulated steps per action may offer an even smoother execution; however, the interpolation would also slow down, while fewer simulated steps per action would increase the speed but decrease the smoothness of the executed gestures.

The simulated version of the robot offers a more efficient testing environment than the real-world version, providing no risk to potentially damage the real robot and allowing for more inventive and creative experimentation. The model used a padding feature to repeat the last frame of each sequence. Usually this is used to lengthen sequences that are not uniform with the rest or are not long enough; however, in the dataset which was used, all of the sequences had a uniform number of frames. In this case, the padding was done to produce a stable final-state feature so that the model is able to recognize the static end of each gesture allowing it to know when each gesture has ended. The normalization and padding done for the test set was required to match the training set as any differences between them could shift SoftMax outputs and cause label flips and mismatches. Additionally, the gestures must have the same indexes in testing as they do in training to not confuse the model. Although misaligned indexes lead the model to predict the correct gesture, they cause the robot to map this prediction to a different gesture.

5.3 Limitations

It is critical to reflect on the limitations of the project, as no method is perfect.

The main limitation on the project was in relation to the data. From the dataset, 3972 clips were used in total, 662 per gesture across 6 gestures. From the total, 3000 clips in total or 500 per gesture were used for training and validation, and 972 clips in total or 162 per gesture were used for testing. When comparing to state-of-the-art deep learning datasets, the Depth_Camera_Dataset is small. To explore more robust LSTM-RNN architectures, a dataset containing tens of thousands of clips would provide much better groundwork for the models to generalize and avoid overfitting. In the Depth_Camera_Dataset, there are also only six gesture classes. For applications in real world interfaces, dozens of gestures are often required which would also require a larger dataset. In addition to this, there is an underrepresentation of speed and angle variations within each gesture class of the dataset, leaving the model potentially unequipped with generalization to novel variations.

The dataset also contains a narrow range of diversity in the subjects. Age, gender, and skin tone are not represented in equal amounts so the model will not generalize as well to the less represented or unrepresented demographics. The lighting and backgrounds lack the necessary diversity required for real-world applications as well, which harms the generalization of the model, although this can be partially circumvented by cropping the region of interest to just the hand using MediaPipe's hand landmark API. The structure of the dataset also lends itself to being subject-disjoint, as there is no way to tell which subject is in which sequence without looking at the images in each sequence. Due to this, the same subject, but not the same gesture sequence, is likely to wind up in both the training and test set. Another limitation arising from the dataset is that there was no raw depth data, but only depth visualizations which were RGB encoded. This meant that the model had to learn from the visualizations of the depth data. To learn the depth modality, the model was taught to estimate the depth data based on the RGB visualizations of the depth, decoding the data back to raw depth data. There is some concern for accuracy here as these are simply estimations of the depth, calculated by working backwards from when the data was encoded into RGB visualizations. The limitation here is that the dataset does not provide information about how the depth data was encoded.

Regarding the useability of the model for real-world real-time video applications, there are several limitations. The restriction of 40 frames per video for each gesture causes problems when attempting to use real-time video as an input, as often the gesture will not be performed exactly

over 40 frames in real-time. If the gesture is performed in real-time through a live video feed, the speed at which the gesture is performed would affect the classification of the gesture. Performing the gesture too slowly or quickly could cause the model to misclassify it. This may cause the model to perform poorly when faced with real-time real-world videos as inputs. When dealing with real-time pipelines, certain limitations may occur from frame rate mismatch or a camera with a particularly low frame rate. With the model being trained to have a certain amount of frames per sequence, and each of the frames containing temporal information that indicates the way in which a dynamic gesture is moving over time, it will not be able to generalize as well to new input data which that is inconsistent with the input data used in training and testing. Using the same Intel RealSense D435 camera would likely not cause this kind of problem to arise; however, for applications in which a camera with different qualities such as a smaller size or lower cost are required, the data gathered by said cameras could cause difficulties for the model when trying to generalize. Additionally, to use the model in a real-world application, a seventh gesture class would need to be used to represent the cases where no gesture was detected. A non-gesture class may be the most commonly classified gesture, thus making the classification of the six gestures more difficult.

Although the vision-based approaches circumvent the need for any sensors being attached to the users, there is still a limitation of the distance between the user and the camera, regardless of the type of camera. Additionally, lighting and the angle between the camera and hand are also limitations for vision-based approaches, both in the creation of the datasets and in the case of applications that use live inputs.

In terms of computation considerations, the training requires GPU-accelerated high-performance computers such as NVIDIA V100. The addition of more gestures, modalities, or simply large scaling of the current gestures would slow down the pipeline.

The simulated Franka Emika Panda robot used the PyBullet physics engine in Python. Although PyBullet offers a simulated environment that is meant to closely resemble the real-world, it is important to note that the dynamics of the robot such as joint trajectories, torque and impedance characteristics, and the gripper mechanics do not fully reflect the hardware, influencing timing and poses when ported to the real non-simulated Franka Emika Panda robot. Although the robot has been designed with affordability and accessibility in mind as mentioned in the designer's ideals, it

was not available to use in this work. The open source simulated version offers a much more affordable and accessible version of the robot, albeit without the luxury of being able to test it in the real world.

The operation of the whole system in this work is dependent on the preprocessing of the dataset, limiting its use on new real-world inputs. It assumes consistency in normalization, padding, and frame ordering between training, Comma-Separated Values (CSV) generation, testing, and robotic control, allowing for an effective model for the test set, but potentially problematic if new input data is collected and used. Although it was possible to design joint trajectories for the Franka Emika Panda robot to mimic closely the human performed gestures in the dataset, the simulation environment and the chosen simulated robot constrict the possible motion of the robot. For example, the scroll left and scroll right gestures require the robotic arm to be configured in a way in which the motion of the end effector must be curved due to the revolute joints. Additionally for the zoom in and zoom out gestures, the robotic arm's end effector must follow a curved trajectory to achieve a similar motion to how a human would perform the gesture, as the robot's joints are not prismatic. Although humans do not possess the capability to move their arms like a prismatic joint, the kind of movement possible with these joints could allow for straighter trajectory lines. A real world robot or different simulated robot would possibly be capable of providing an even more specific set of joint trajectories that more closely resemble the human performed gestures.

5.4 Future Work

There are several ways to further develop this project in the future, as observed in the limitations. The most obvious is to increase the number of gesture classes by adding dynamic gestures such as pinching, waving, etc. Additionally, the subject pool for the dataset should be increased to include subjects across ages, genders, hand sizes, skin tones, and handedness. The dataset could be expanded by concatenating other similar datasets, combining into one larger dataset. There are issues with compatibility that would need to be resolved in such a scenario. The model that was trained on one dataset could be tested on other datasets without retraining to observe the generalization when faced with novel data. Audio may also be a novel modality to integrate as it opens the door to exploring audio-visual cues for gestures that have accompanying speech or sounds.

To address the problem of the algorithm not being able to distinguish between similar gestures in Hax et al. (2024), the algorithm can be trained to recognize the starting point for each gesture in addition to the full gesture to be classified. This may be achieved through observation of the velocity of the gestures, or by splitting gestures into different segments and classifying which movement in the sequence is the most relevant to the gesture and can be identified with the highest probability. People make slight movements before, while, or after performing a gesture which can make it difficult for the model to classify the gesture.

Fine-tuning the InceptionV3 pre-trained model and creating a truly end-to-end pipeline may also provide better results. Rather than using offline feature extraction and freezing the InceptionV3 CNN, allowing the model to fully backpropagate could lead to significant improvements in the model's potential for real-world applications. Additionally, feature fusion may be experimented with at different times in the pipeline as well, instead of the model-level feature fusion used in this work. This could include allowing the models to go fully through the pipelines and using decision-level feature fusion, or early-level feature fusion where the features are concatenated earlier before going through the pipeline.

Fusion of PointLSTM and DepthCRNN could outperform regular PointLSTM. Future work may involve fusing quantized depth images with more modalities such as point clouds, skeleton coordinates, and motion features. The performance may further improve with other modalities, particularly privacy preserving modalities like thermal images. Additionally, depth images can be fed straight into the CNN without any sort of conversion into raw depth data as InceptionV3 could be able to handle this approach by incorporating a separate depth channel alongside the RGB channel. In this project, an LSTM-RNN architecture was used to observe the impacts of fusing the depth modality with the RGB modality in the testing of the model. However, the state-of-the-art method for deep learning visual based pipelines is to use transformers such as temporal transformers or ViT models. Such approaches could be explored.

A practical significance of this study is in the DHGR application of real-time simulated robotic hand control. For instance, Gourob et al. (2021) demonstrates real-time static hand gesture recognition to control a robotic hand's fingers. Adapting this approach to dynamic hand gestures can unlock a broader range of motions and more natural human-robot interaction (HRI), especially when integrated with a robotic arm which includes an elbow joint.

The torque and impedance control in PyBullet or a higher fidelity simulator may be used to evaluate the motions of the robot and reduce the gap between simulation and hardware. The simulated robot has a two-finger gripper end effector which can be used to represent the simple gestures in the dataset used for this project. However, for more complex dynamic gestures, a more realistic humanoid hand with all five fingers may be required. For systems intending to use the HGR model for complex dynamic hand gesture datasets, a different robot may be more beneficial, such as the one developed by Miyama et al. (2025). A whole-body humanoid robot that includes arms and hands with intricate mechanics may allow for additional possible applications of the HGR system. To extend the end-to-end implementation of the system, the discrete labels being fed to explicit joint trajectories may be replaced by a model that directly predicts joint trajectories from the precomputed features, or a combination of the model predicting the joint trajectories from the precomputed features with maintaining some explicit joint trajectories that are hard set by the user.

Future work can be done on developing real-time DHGR systems, where the model receives new inputs from a user who is performing one of the six gestures in front of the camera, the model performs the proper preprocessing, predicts the gesture, and sends the command to the simulated robot to perform with very little delay. Using the robotics control aspect, the six gestures from the dataset may be translated into any arbitrary set of commands for the simulated robot. In this work, although the robot is designed to mimic the movements of the gestures in the dataset, it could also be designed to perform simpler or more complex gestures. There are several possible applications that the model can be used for, for example to potentially enhance the usability of robots in various fields, such as assistive technology, teleoperation, and virtual reality. Fine control of robotic gestures would allow applications in more serious situations that require expert precision but that are not in suitable or safe environments for humans.

6. Conclusion

This thesis presents a dynamic hand gesture recognition (DHGR) system, expanding on previous works that included just the RGB modality by incorporating the depth modality as well in multi-modal fusion. With this novel approach, the model's performance in training and testing was improved

By training the model on full sequences, using a sequence-based, stratified 5-fold cross-validation method, the gestures' temporal information is preserved. The sequence-based data split approach was crucial in allowing the model to learn and validate the temporal dynamics in hand gestures effectively, leading to higher accuracy and more robust recognition.

The multi-modal feature extraction pipeline fuses spatial features from parallel RGB and depth InceptionV3 streams with the 2.5D hand-landmark coordinates. These are fed into a stacked LSTM classifier. The full-sequence training improved testing accuracy compared to the state-of-the-art published results, improving the model's ability to distinguish between similar motions such as zoom in and zoom out. In comparison to the previous InceptionV3-LSTM studies on the same dataset we used, the model in this work performed 5.1% better than the Yaseen et al. (2024) study and 11.2% better than the Hax et al. (2024) study.

The work in this thesis shows clear paths to practical improvements and real-world applications. Incorporating some form of start/end detector would reduce errors caused by the model's inability to understand pre-gesture motions. Transformer-based temporal encoders may be used instead of stacked LSTMs to yield further gains in accurately differentiating similar looking gestures. By extending the current pipeline, a real-time DHGR system may be developed that can be deployed for human-robot interaction.

7. References

Allocations and Compute Scheduling. Allocations and compute scheduling - Alliance Doc. (n.d.). https://docs.alliancecan.ca/wiki/Allocations_and_compute_scheduling

Almuqati, M. T., Sidi, F., Rum, S. N., Zolkepli, M., & Ishak, I. (2024). Challenges in supervised and unsupervised learning: a comprehensive overview. *International Journal on Advanced Science, Engineering and Information Technology*, *14*(4), 1449–1455. https://doi.org/10.18517/ijaseit.14.4.20191

Andrushchenko, M., Selivanova, K., Avrunin, O., Palii, D., Tymchyk, S., & Turlykozhayeva, D. (2024). Hand movement disorders tracking by smartphone based on Computer Vision Methods. *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, *14*(2), 5–10. https://doi.org/10.35784/iapgos.6126

Balaji, P., & Prusty, M. R. (2024). Multimodal fusion hierarchical self-attention network for dynamic hand gesture recognition. *Journal of Visual Communication and Image Representation*, *98*, 104019. https://doi.org/10.1016/j.jvcir.2023.104019

Chatterjee, K., Raju, M., Selvamuthukumaran, N., Pramod, M., Kumar, B. K., Bandyopadhyay, A., & Mallik, S. (2024). Hack: Hand gesture classification using a convolutional neural network and generative adversarial network-based Data Generation Model. *Information*, *15*(2), 85. https://doi.org/10.3390/info15020085

Chevtchenko, S. F., Vale, R. F., & Macario, V. (2018). Multi-objective optimization for hand posture recognition. *Expert Systems with Applications*, *92*, 170–181. https://doi.org/10.1016/j.eswa.2017.09.046

Coumans, E., & Bai, Y. (2016). Pybullet, a python module for physics simulation for games, robotics and machine learning

Cruz, C. A., & Igarashi, T. (2020). A survey on interactive reinforcement learning. *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, 1195–1209. https://doi.org/10.1145/3357236.3395525

Csonka, G., Khalid, M., Rafiq, H., & Ali, Y. (2023). AI-based hand gesture recognition through camera on robot. *2023 International Conference on Frontiers of Information Technology (FIT)*, 256–261. https://doi.org/10.1109/fit60620.2023.00054

De Smedt, Q., Wannous, H., & Vandeborre, J. P. (2016). Skeleton-based dynamic hand gesture recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 1206–1214. https://doi.org/10.1109/cvprw.2016.153

Deng, Z., Leng, Y., Chen, J., Yu, X., Zhang, Y., & Gao, Q. (2024). TMS-net: A multi-feature multi-stream multi-level Information Sharing Network for skeleton-based Sign language recognition. *Neurocomputing*, *572*, 127194. https://doi.org/10.1016/j.neucom.2023.127194

Ding, Y., Pang, H., Wu, X., & Lan, J. (2011). Static hand-gesture recognition using hog and improved LBP features. *International Journal of Digital Content Technology and Its Applications*, *5*(11), 236–243. https://doi.org/10.4156/jdcta.vol5.issue11.30

Dominio, F., Donadeo, M., & Zanuttigh, P. (2014). Combining multiple depth-based descriptors for hand gesture recognition. *Pattern Recognition Letters*, *50*, 101–111. https://doi.org/10.1016/j.patrec.2013.10.010

Dutta, M., Gupta, D., Gulzar, Y., Mir, M. S., Onn, C. W., & Soomro, A. B. (2024). Leveraging inception V3 for precise early and late blight disease classification in potato crops. *Traitement Du Signal*, *41*(2), 705–715. https://doi.org/10.18280/ts.410213

Elouariachi, I., Benouini, R., Zenkouar, K., & Zarghili, A. (2020). Robust hand gesture recognition system based on a new set of quaternion Tchebichef moment invariants. *Pattern Analysis and Applications*, *23*(3), 1337–1353. https://doi.org/10.1007/s10044-020-00866-9

Feng, K., & Yuan, F. (2013). Static hand gesture recognition based on hog characters and support Vector Machines. *2013 2nd International Symposium on Instrumentation and Measurement, Sensor Network and Automation (IMSNA)*, 936–938. https://doi.org/10.1109/imsna.2013.6743432

Gallouédec, Q., Cazin, N., Dellandréa, E., & Chen, L. (2021). In *Panda-Gym: Open-source goal-conditioned environments for robotic learning*.

Garg, M., Ghosh, D., & Pradhan, P. M. (2024). Gestformer: Multiscale Wavelet Pooling Transformer Network for dynamic hand gesture recognition. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2473–2483. https://doi.org/10.1109/cvprw63382.2024.00254

Gourob, J. H., Raxit, S., & Hasan, A. (2021). A robotic hand: Controlled with vision based hand gesture recognition system. *2021 International Conference on Automation, Control and Mechatronics for Industry 4.0 (ACMI)*, 1–4. https://doi.org/10.1109/acmi53878.2021.9528192

Haddadin, S. (2024). The Franka Emika Robot: A standard platform in robotics research. *IEEE Robotics & Automation Magazine*, *31*(4), 136–148. https://doi.org/10.1109/mra.2024.3451788

Hax, D. R., Penava, P., Krodel, S., Razova, L., & Buettner, R. (2024). A novel hybrid deep learning architecture for dynamic hand gesture recognition. *IEEE Access*, *12*, 28761–28774. https://doi.org/10.1109/access.2024.3365274

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. https://doi.org/10.1109/cvpr.2016.90

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hou, J., Wang, G., Chen, X., Xue, J. H., Zhu, R., & Yang, H. (2019). Spatial-temporal attention RES-TCN for skeleton-based dynamic hand gesture recognition. *Lecture Notes in Computer Science*, 273–286. https://doi.org/10.1007/978-3-030-11024-6_18

Jeeru, S., Sivapuram, A. K., León, D. G., Gröli, J., Yeduri, S. R., & Cenkeramaddi, L. R. (2022). Depth camera based dataset of hand gestures. *Data in Brief*, *45*, 108659. https://doi.org/10.1016/j.dib.2022.108659

John, V., Boyali, A., Mita, S., Imanishi, M., & Sanma, N. (2016). Deep learning-based fast hand gesture recognition using representative frames. *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 1–8. https://doi.org/10.1109/dicta.2016.7797030

Keras Documentation: Inceptionv3. Keras.io. (n.d). https://keras.io/api/applications/inceptionv3/

Koca, O., Kaymakci, O. T., & Mercimek, M. (2020). Advanced predictive maintenance with machine learning failure estimation in Industrial Packaging Robots. *2020 International Conference on Development and Application Systems (DAS)*, 1–6. https://doi.org/10.1109/das49615.2020.9108913

Kopuklu, O., Gunduz, A., Kose, N., & Rigoll, G. (2020). Online dynamic hand gesture recognition including efficiency analysis. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, *2*(2), 85–97. https://doi.org/10.1109/tbiom.2020.2968216

Leon, D. G., Groli, J., Yeduri, S. R., Rossier, D., Mosqueron, R., Pandey, O. J., & Cenkeramaddi, L. R. (2022). Video hand gestures recognition using depth camera and lightweight CNN. *IEEE Sensors Journal*, *22*(14), 14610–14619. https://doi.org/10.1109/jsen.2022.3181518

Li, Y., & Zhang, P. (2022). Static hand gesture recognition based on hierarchical decision and classification of finger features. *Science Progress*, *105*(1).

Linkon, A. H., Labib, Md. M., Hasan, T., Hossain, M., & Jannat, M. E. (2021). Deep learning in prostate cancer diagnosis and Gleason grading in histopathology images: An extensive

study. *Informatics in Medicine Unlocked*, *24*, 100582. https://doi.org/10.1016/j.imu.2021.100582

Liu, Y., Li, X., & Yang, L. (2024). A wearable sensor-based dynamic gesture recognition model via broad attention learning. *Signal, Image and Video Processing*, *19*(1). https://doi.org/10.1007/s11760-024-03567-6

López-Casado, C., Bauzano, E., Rivas-Blanco, I., Pérez-del-Pulgar, C. J., & Muñoz, V. F. (2019). A gesture recognition algorithm for hand-assisted laparoscopic surgery. *Sensors*, *19*(23), 5182. https://doi.org/10.3390/s19235182

Lu, Z., Qin, S., Lv, P., Sun, L., & Tang, B. (2023). Real-time continuous detection and recognition of dynamic hand gestures in untrimmed sequences based on end-to-end architecture with 3D DenseNet and LSTM. *Multimedia Tools and Applications*, *83*(6), 16275–16312. https://doi.org/10.1007/s11042-023-16130-1

Mahmud, H., Morshed, M. M., & Hasan, Md. K. (2023). Quantized depth image and skeleton-based multimodal dynamic hand gesture recognition. *The Visual Computer*, *40*(1), 11–25. https://doi.org/10.1007/s00371-022-02762-1

Manganaro, F., Pini, S., Borghi, G., Vezzani, R., Cucchiara, R. (2019). Hand Gestures for the Human-Car Interaction: the Briareo dataset. *20th International Conference on Image Analysis and Processing Trento (ICIAP), 560-571.* https://doi.org/10.1007/978-3-030-30645-8_51

Marin, G., Dominio, F., & Zanuttigh, P. (2015). Hand gesture recognition with jointly calibrated leap motion and depth sensor. *Multimedia Tools and Applications*, *75*(22), 14991–15015. https://doi.org/10.1007/s11042-015-2451-6

Miyama, K., Kawaharazuka, K., Okada, K., & Inaba, M. (2023). Development of a five-fingerd biomimetic soft robotic hand by 3D printing the skin and skeleton as one unit. *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6624–6630. https://doi.org/10.1109/iros55552.2023.10341570

Molchanov, P., Yang, X., Gupta, S., Kim, K., Tyree, S., & Kautz, J. (2016). Online detection and classification of dynamic hand gestures with recurrent 3D convolutional neural networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4207–4215. https://doi.org/10.1109/cvpr.2016.456

Naik, K. J., & Soni, A. (2021). Video classification using 3D Convolutional Neural Network. *Advances in Information Security, Privacy, and Ethics*, 1–18. https://doi.org/10.4018/978-1-7998-2795-5.ch001

Oliveira, M., Chatbri, H., Little, S., O'Connor, N. E., & Sutherland, A. (2017). A comparison between end-to-end approaches and feature extraction based approaches for sign language recognition. *2017 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, 1–6. https://doi.org/10.1109/ivcnz.2017.8402478

Oudah, M., Al-Naji, A., & Chahl, J. (2020). Hand gesture recognition based on Computer Vision: A review of techniques. *Journal of Imaging*, *6*(8), 73. https://doi.org/10.3390/jimaging6080073

Oyedotun, O. K., & Khashman, A. (2016). Deep learning in vision-based static hand gesture recognition. *Neural Computing and Applications*, *28*(12), 3941–3951. https://doi.org/10.1007/s00521-016-2294-8

Park, J., Choi, H., & Kim, J. (2014). Hand pose recognition by using masked zernike moments. *Proceedings of the 9th International Conference on Computer Vision Theory and Applications*, 551–556. https://doi.org/10.5220/0004731605510556

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research, 12, 2825–2830.*

Priyal, S. P., & Bora, P. K. (2013). A robust static hand gesture recognition system using geometry based normalizations and Krawtchouk Moments. *Pattern Recognition*, *46*(8), 2202–2219. https://doi.org/10.1016/j.patcog.2013.01.033

Rastgoo, R., Kiani, K., Escalera, S., & Sabokrou, M. (2024). Multi-modal zero-shot dynamic hand gesture recognition. *Expert Systems with Applications*, *247*, 123349. https://doi.org/10.1016/j.eswa.2024.123349

RGB cameras: Definition, components, and integration. TechNexion. (2025). https://www.technexion.com/resources/rgb-cameras/

Rogel, A., Savery, R., Yang, N., & Weinberg, G. (2022). Robogroove: Creating fluid motion for Dancing Robotic Arms. *Proceedings of the 8th International Conference on Movement and Computing*, 1–9. https://doi.org/10.1145/3537972.3537985

Sahoo, J. P., Prakash, A. J., Pławiak, P., & Samantray, S. (2022). Real-time hand gesture recognition using fine-tuned convolutional neural network. *Sensors*, *22*(3), 706. https://doi.org/10.3390/s22030706

Samaan, G. H., Wadie, A. R., Attia, A. K., Asaad, A. M., Kamel, A. E., Slim, S. O., Abdallah, M. S., & Cho, Y.-I. (2022). MediaPipe's landmarks with RNN for dynamic sign language recognition. *Electronics*, *11*(19), 3228. https://doi.org/10.3390/electronics11193228

Sanga, S. L., Machuve, D., Mero, V., & Mwanganda, D. (2020). Mobile-based deep learning models for banana disease detection. *Engineering, Technology & Applied Science Research*, *10*(3), 5674–5677. https://doi.org/10.48084/etasr.3452

Satybaldina, D., & Kalymova, G. (2021). Deep learning based static hand gesture recognition. *Indonesian Journal of Electrical Engineering and Computer Science*, *21*(1), 398. https://doi.org/10.11591/ijeecs.v21.i1.pp398-405

Sebai, D. (2024). *Depth in focus: D component of RGB-D images and videos*. Springer Nature Research Communities. https://communities.springernature.com/posts/depth-in-focus-d-component-of-rgb-d-images-and-videos

Sebbah, R., & Chelali, F. Z. (2024). IWBC and LFD for static and dynamic hand gesture recognition. *2024 8th International Conference on Image and Signal Processing and Their Applications (ISPA)*, 1–7. https://doi.org/10.1109/ispa59904.2024.10536841

Staudemeyer, R. C., & Morris, E. R. (2019). Understanding LSTM – A tutorial into Long Short-Term Memory Recurrent Neural Networks. https://doi.org/10.48550/arXiv.1909.09586

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2818–2826. https://doi.org/10.1109/cvpr.2016.308

TensorFlow Basics: TensorFlow Core. TensorFlow. (2024). https://www.tensorflow.org/guide/basics

TensorFlow Keras: The high-level API for TensorFlow: TensorFlow Core. TensorFlow. (2024). https://www.tensorflow.org/guide/keras

TensorFlow Mixed precision: TensorFlow Core. TensorFlow. (2024). https://www.tensorflow.org/guide/mixed_precision

Tripathi, R., & Verma, B. (2023). Survey on vision-based dynamic hand gesture recognition. *The Visual Computer*, *40*(9), 6171–6199. https://doi.org/10.1007/s00371-023-03160-x

Wang, X., Liu, Y., SUN, C., Wang, B., & Wang, X. (2015). Predicting polarities of tweets by composing word embeddings with long short-term memory. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. https://doi.org/10.3115/v1/p15-1130

Xie, J., Xu, Z., Zeng, J., Gao, Y., & Hashimoto, K. (2025). Human–robot interaction using dynamic hand gesture for teleoperation of quadruped robots with a robotic arm. *Electronics*, *14*(5), 860. https://doi.org/10.3390/electronics14050860

Yaseen, Kwon, O.-J., Kim, J., Jamil, S., Lee, J., & Ullah, F. (2024). Next-Gen Dynamic Hand Gesture Recognition: MediaPipe, inception-V3 and LSTM-based enhanced deep learning model. *Electronics*, *13*(16), 3233. https://doi.org/10.3390/electronics13163233

Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C.-L., & Grundmann, M. (2020). *MediaPipe hands: On-device real-time hand tracking*. arXiv.org. https://arxiv.org/abs/2006.10214

Zhi, D. (2018). *Depth camera-based hand gesture recognition for training a robot to perform sign language* (thesis). *Depth Camera-Based Hand Gesture Recognition for Training a Robot to Perform Sign Language*. Université d'Ottawa / University of Ottawa, Ottawa, Ontario.

Zhu, Q. (2020). *Teleoperated grasping using an upgraded haptic-enabled human-like robotic hand and a CyberTouch glove* (thesis). *Teleoperated Grasping Using an Upgraded Haptic-Enabled Human-Like Robotic Hand and a CyberTouch Glove*. Université d'Ottawa / University of Ottawa, Ottawa, Ontario.

Zinser, M., Rose, J., & Sirkin, H. (2025). *The robotics revolution: The next great leap in manufacturing*. BCG Global. https://www.bcg.com/publications/2015/lean-manufacturing-innovation-robotics-revolution-next-great-leap-manufacturing