Three-Dimensional Hand Gestures Classification: Comparative Study of Machine Learning Techniques

GROUP 10

Bousmar Cyril - 63401800
UCLouvain, EPL - SINF
cyril.bousmar@uclouvain.be

Delsart Mathis - 31302100
UCLouvain, EPL - INFO
mathis.delsart@student.uclouvain.be

Lamien Sienou - 72642301
UCLouvain, EPL - SINF
sienou.lamien@student.uclouvain.be

I. Introduction

Gesture recognition is a key component in human-computer interaction, enabling intuitive and natural communication between users and machines. With the rise of spatial computing and wearable devices, the ability to accurately classify 3D hand-drawn gestures has become increasingly relevant. This project investigates the performance of various classification models for recognizing such gestures, using two publicly available datasets introduced by Huang et al. [1].

To address the task of 3D gesture classification, a range of models was explored, spanning from simple baseline methods (such as k-nearest neighbors with different distance metrics), as well as more advanced techniques such as the $1 recognizer, Hidden Markov Models, and deep learning approaches incorporating feature extraction. Each model was evaluated under two complementary validation schemes: a user-independent setting to assess generalization to unseen users, and a user-dependent setting to evaluate performance when user identity is available during training.

The objective of this study is to systematically compare these methods across both datasets and validation protocols, identify their respective strengths and limitations, and gain insights into the challenges inherent to 3D gesture recognition. Model performance is assessed quantitatively and qualitatively using accuracy, standard deviation, and confusion matrices.

This report is structured as follows: Section II introduces the datasets used in this study. Section III describes the compared models, categorized into baseline and advanced approaches. Section IV details the methodology employed to evaluate model performance. Section V presents the experimental results, and Section VI provides an interpretation of the findings along with directions for future research.

II. Datasets

This study utilizes two publicly available gesture datasets, originally introduced by Huang et al. (2019) [1]. Both datasets were collected under the same experimental protocol, involving ten participants, but differ in the types of gestures and their associated labels.

The first dataset, referred to as ***domain1***, consists of 3D hand-drawn representations of the digits 0 through 9. Each participant drew each digit ten times, resulting in a total of 1,000 gesture sequences. These digit gestures are relatively simple and often representable in two dimensions, making *domain1* an appropriate starting point for model development and evaluation.

The second dataset, referred to as ***domain4***, includes 3D hand-drawn geometric figures such as pyramids and other complex shapes. Similar to *domain1*, each participant drew each figure ten times. However, unlike the digit gestures, these shapes generally require the full three-dimensional space for accurate representation, introducing greater spatial and structural complexity.

Due to this increased dimensionality and variability, *domain4* presents a more challenging classification task. As such, it serves as a valuable benchmark for evaluating a model's ability to generalize to more complex gesture patterns.

While this study focuses primarily on *domain1* and *domain4*, additional gesture datasets from the original collection are available and may be explored in future work for further analysis or benchmarking.

III. Models

This section describes the models used for gesture classification, categorized into two groups: baseline models, which serve as benchmarks due to their simplicity, and advanced models, which are expected to offer superior performance due to their increased complexity.

It is important to note that, with the exception of the *KNN Classifier - DTW Distance* model, all models assume constant time steps between gesture samples, as outlined in the project specifications.

A. Baseline

1) KNN Classifier - Euclidean Distance

The K-Nearest Neighbors (KNN) classifier using Euclidean distance is one of the most straightforward method for classification. In this approach, each gesture is represented as a point in a multidimensional feature space. The classifier assigns a label to a test gesture based on the majority class among its k nearest neighbors from the training set.

The Euclidean distance between two gesture vectors is computed as:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^{M} (x_{im} - x_{jm})^2} \tag{1}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the feature vectors of two gestures, and M is the number of features.

This model operates under the assumption that gestures which are spatially close in the feature space are more likely to belong to the same class.

*2) **KNN Classifier - DTW Distance***

The K-Nearest Neighbors (KNN) classifier with Dynamic Time Warping (DTW) distance extends the standard KNN approach to handle time-series data, such as gesture trajectories. Unlike fixed-length vector comparisons, DTW allows for elastic shifting of the time axis, making it well-suited for gestures that may vary in duration or speed.

The DTW distance between two time-series sequences $\mathbf{X} = \{x_1, x_2, \ldots, x_T\}$ and $\mathbf{Y} = \{y_1, y_2, \ldots, y_T\}$ is defined as:

$$d_{DTW}(\mathbf{X}, \mathbf{Y}) = \min_{\pi \in \Pi} \sum_{t=1}^{T} d(x_t, y_{\pi(t)}) \tag{2}$$

where Π denotes the set of all valid warping paths, and $d(x_t, y_{\pi(t)})$ is a local distance measure (Euclidean in our implementation) between aligned time steps. The warping path π determines how elements from the two sequences are matched.

*3) **KNN Classifier - Edit Distance***

The K-Nearest Neighbors (KNN) classifier with Edit Distance is applied when gestures are represented as sequences of discrete symbols or actions. Edit Distance (also known as Levenshtein Distance) quantifies the similarity between two sequences by computing the minimum number of operations (insertions, deletions, or substitutions) required to transform one sequence into the other.

Using dynamic programming, the Edit Distance is computed as follows:

Initialization:

$$D^*(x_0^{|x|}, y_0^0) = 0$$

Recursive formulation:

$$D^*(x_i^{|x|}, y_0^j) = \min \begin{cases} D^*(x_i^{|x|}, y_0^{j-1}) + 1 & \text{(insertion)} \\ D^*(x_{i-1}^{|x|}, y_0^j) + 1 & \text{(deletion)} \\ D^*(x_{i-1}^{|x|}, y_0^{j-1}) + \delta_{ij} & \text{(substitution)} \end{cases}$$

where $\mathbf{x}$ and $\mathbf{y}$ are gesture strings of lengths $|x|$ and $|y|$ respectively, $x_i^{|x|}$ and y_0^j denote prefixes of length i and j, and $\delta_{ij} = 0$ if $x_i = y_j$, and 1 otherwise.

Final result:

$$d_{\text{Edit}}(\mathbf{x}, \mathbf{y}) = D^*(x_{|x|}^{|x|}, y_0^{|y|}) \tag{3}$$

This method is particularly effective when gestures consist of discrete steps or symbolic representations (e.g., segmented strokes). The classifier labels a test gesture by identifying the training gesture with the smallest edit distance.

To apply this method, gestures must first be converted into symbolic sequences. This is achieved through a preprocessing step using the **k-means clustering** algorithm, in which gesture data points are assigned to one of 10 centroids per domain. Each centroid is mapped to a unique symbol (e.g., a letter), and a gesture is represented as a string by concatenating the symbols corresponding to its constituent points.

*4) **KNN Classifier – Longest Common Subsequence (LCS) Distance***

The K-Nearest Neighbors (KNN) classifier with Longest Common Subsequence (LCS) distance measures the similarity between two sequences by identifying the longest subsequence that appears in both sequences in the same order, without requiring elements to be contiguous. This method captures structural similarities while being tolerant to noise and variations in gesture execution.

As in the KNN with Edit Distance, gestures are first symbolized using the same preprocessing pipeline: applying k-means clustering with 10 centroids per domain, assigning each point to a representative symbol, and converting gesture sequences into symbolic strings.

Unlike the edit distance, LCS does not permit substitutions (only insertions and deletions) when aligning sequences. Based on the formulation by Bergroth [4], the LCS length can be derived from the edit distance as:

$$d_{\text{LCS}}(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \left(|\mathbf{x}| + |\mathbf{y}| - d_{\text{Edit}}(\mathbf{x}, \mathbf{y}) \right) \tag{4}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the symbolic representations of two gestures, $|\mathbf{x}|$ and $|\mathbf{y}|$ denote their lengths, and d_{Edit} is the standard edit distance.

The resulting LCS value serves as a similarity metric: the longer the common subsequence, the more similar the gestures. The classifier assigns the label of the most similar gesture in the training set based on this distance.

B. Advanced

*1) **Long Short-Term Memory (LSTM)***

The Long Short-Term Memory (LSTM) model is well-suited for sequential data such as 3D gesture trajectories, due to its ability to capture long-term temporal dependencies. It processes sequences step-by-step while maintaining an internal state, making it effective for modeling dynamic gesture patterns.

Model Architecture:

- *Layer 1 – LSTM (128 units):* Captures temporal dependencies across the input sequence while producing a sequence of outputs.
- *Layer 2 – Dropout (rate: 0.5):* Prevents overfitting by randomly deactivating 50% of the neurons during training.
- *Layer 3 – LSTM (64 units):* Further compresses the temporal features for dimensionality reduction.
- *Layer 4 – Dropout (rate: 0.3):* Additional regularization to improve generalization.

- *Layer 5 – Dense (64 units, ReLU):* Fully connected layer for non-linear transformation of learned features.
- *Layer 6 – Output Dense (Softmax):* Final classification layer using softmax activation for multi-class prediction.

The model is trained using the categorical crossentropy loss function, optimized with the Adam optimizer. Accuracy is used as the evaluation metric.

During preprocessing, gesture sequences are normalized and resampled to a fixed number of time steps to ensure consistent input dimensions across all samples.

2) $1 Recognizer

The $1 Recognizer is a simple, fast, and easy-to-implement gesture recognition algorithm, particularly well-suited for use in user interfaces [2]. It is known for its robustness against human variations in drawing gestures and its efficiency in processing gestures in real-time. The algorithm works on 2D gesture data and is designed to be quick to train and classify.

Main Stages of the Algorithm:

- *Resampling:* Gestures are resampled to a uniform length of N points. This is done through linear interpolation along the trajectory of the gesture, ensuring that all gestures have the same number of points, making them easier to compare.
- *Dimensionality Reduction:* The $1 Recognizer operates on 2D data, so the third dimension (z-axis) is ignored. The gesture data is reduced to the x and y axes to work in a 2D space.
- *Orientation Rotation and Scale Normalization:* The first and last points of the gesture are aligned along the horizontal axis (the x-axis) for consistency. An affine transformation is then applied to scale and normalize the gesture coordinates into the range $[0, 1]$, ensuring that all gestures are on a comparable scale and orientation.
- *Similarity Measurement:* The $1 Recognizer uses Procrustes analysis to measure the dissimilarity between a candidate gesture and a template gesture. The objective is to find the optimal alignment between the candidate gesture and the template by minimizing the disparity function:

$$\text{Disparity} = \min_{s,\theta,\mathbf{t}} \sum_{i=1}^{N} \|sR(\theta)\mathbf{c}_i + \mathbf{t} - \mathbf{t}_i\| \tag{5}$$

  where:
  - $\mathbf{c}_i$ is the i-th point of the candidate gesture,
  - $\mathbf{t}_i$ is the corresponding point in the template gesture,
  - s is the scaling factor,
  - $R(\theta)$ is the 2D rotation matrix for alignment,
  - $\mathbf{t}$ is the translation vector, and
  - N is the number of points in the gesture.

  The result of the Procrustes analysis is the transformation parameters that best align the candidate gesture with the template gesture. The lower the disparity, the higher the similarity between the two gestures.

This approach enables the recognition of gestures even with variations in speed, position, or slight distortions, making it ideal for real-time applications in user interfaces where simplicity and speed are key.

3) Logistic Regression

Logistic Regression is employed as a linear classifier trained on a set of statistical features derived from each gesture. These features include position, velocity, and acceleration (calculated from the distance between consecutive points assuming constant time intervals, as specified by the project requirements) as well as orientation-related metrics.

The data is standardized using a Scikit-learn pipeline to ensure consistent feature scaling. The model employs L_1 regularization to promote sparsity, making the learned model more interpretable and resistant to overfitting. Hyperparameters are tuned using Bayesian cross-validation to improve generalization performance.

4) Feedforward Neural Network (FFNN)

The Feedforward Neural Network (FFNN) is trained on the same set of statistical features used in the logistic regression model, including position, velocity, acceleration (assuming constant time steps), and orientation metrics. Unlike logistic regression, the FFNN utilizes a deeper architecture to learn complex, non-linear relationships within the feature space. This approach allows the model to better capture intricate patterns that may be missed by linear classifiers.

Model Architecture:

- *Layer 1 – Dense (256 units, ReLU activation, L2 regularization):* A fully connected layer that processes the input features and extracts high-level patterns. L2 regularization is applied to prevent overfitting.
- *Layer 2 – Batch Normalization:* Normalizes the output of the previous layer, ensuring a stable learning process and improving convergence.
- *Layer 3 – Dropout (rate: 0.4):* A dropout layer with a rate of 40%, which helps in preventing overfitting by randomly deactivating some of the neurons during training.
- *Layer 4 – Dense (128 units, ReLU activation, L2 regularization):* Another fully connected layer that further refines the learned features from the previous layer.
- *Layer 5 – Batch Normalization:* Another batch normalization layer to stabilize and speed up training.
- *Layer 6 – Dropout (rate: 0.4):* Additional dropout for regularization to mitigate overfitting.
- *Layer 7 – Dense (64 units, ReLU activation, L2 regularization):* A final fully connected layer that reduces the dimensionality and prepares the features for the classification step.
- *Layer 8 – Output Dense (Softmax activation):* The output layer uses softmax activation for multi-class classification, assigning probabilities to each gesture class.

The model is trained using the sparse categorical cross-entropy loss function, optimized with the Adam optimizer. The

training includes early stopping based on validation loss and a dynamic learning rate adjustment via a learning rate scheduler.

Gesture sequences are preprocessed by extracting statistical features, which are then standardized before training. This architecture allows the FFNN to effectively capture and classify complex gesture patterns based on the extracted features.

5) Hidden Markov Model (HMM)

The Hidden Markov Model classifier employs a separate Gaussian HMM for each gesture class. In the training process, gesture sequences are first normalized and resampled to a fixed length to ensure consistency across all samples. Additionally, the sequences are scaled to the range of [-1, 1] for numerical stability during model training.

Each Gaussian HMM is trained using the `hmmlearn` library's `GaussianHMM` implementation, which models the temporal dynamics of gesture sequences. The HMM uses a mixture of Gaussians to model the underlying hidden states of each gesture, enabling the capture of both the discrete and continuous characteristics of the sequences.

IV. Methodology

To ensure consistency and reproducibility, all experiments were conducted on a system equipped with an *Apple Silicon M2* chip and 16 GB of RAM.

A. Metrics

The primary evaluation metric used in this study is **classification accuracy**, which quantifies the proportion of correctly classified gestures relative to the total number of gestures in the test set. Formally, given a test set containing n gesture instances, accuracy is defined as:

$$\text{Accuracy} = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(y_i = \hat{y}_i) \tag{6}$$

where y_i denotes the true class label of the i^{th} gesture, $\hat{y}_i$ denotes the predicted class label, and $\mathbb{I}(\cdot)$ is the indicator function, which returns 1 if its argument is true and 0 otherwise.

Accuracy is a suitable metric in this context because the dataset is perfectly balanced across gesture classes, ensuring that each class contributes equally to the final score. In cases of imbalanced class distributions, alternative metrics such as the balanced classification rate (BCR) or F1-score would be preferred to provide a more nuanced evaluation.

To complement the accuracy metric, **confusion matrices** were also analyzed. The confusion matrix C is a square matrix where each entry C_{ij} indicates the number of instances of class i that were predicted as class j. This allows for a detailed examination of model performance, highlighting specific gesture classes that are prone to misclassification and revealing potential asymmetries or biases in the classifier.

B. Validation and Comparison

To comprehensively assess the performance and generalizability of the proposed models, two cross-validation strategies were employed: one **user-independent** and one **user-dependent**. These complementary evaluation protocols ensure that models are tested both on previously unseen users and on new gesture samples from known users.

1) **User-Independent Validation**: This validation setting follows a leave-one-user-out cross-validation protocol. Each fold involves holding out all gesture data from one user (i.e., 10% of the dataset) as the test set, while training the model on the remaining nine users (90%). This process is repeated for each of the ten users, resulting in ten folds. This setting evaluates the model's ability to generalize to new users not present in the training data.
2) **User-Dependent Validation**: In this setting, a leave-one-sample-per-gesture-out cross-validation is applied individually to each user. For every user, one sample of each gesture class (i.e., one out of ten for each digit or shape) is held out for testing, while the remaining nine samples per class are used for training. This process is repeated ten times, such that each sample serves exactly once as a test instance. Unlike the user-independent setup, this protocol evaluates the model's ability to distinguish gesture classes when the user is already represented in the training data.

For both validation strategies, the average classification accuracy and standard deviation were reported to quantify model performance. Additionally, confusion matrices were analyzed to identify gesture classes with frequent misclassifications, enabling a more nuanced understanding of model behavior, strengths, and weaknesses.

V. Results

TABLE I: Comparison of model accuracies (%) under both validation settings for Domains 1 and 4. Higher is better.

Method	**Domain 1**		**Domain 4**	
	U-dep.	**U-indep.**	**U-dep.**	**U-indep.**
KNN - Euclidean	80.2±12.2	18.8±13.6	72.8±11.8	19.7±9.92
KNN - DTW	98.3±1.35	64.3±16.7	**96.9±2.98**	55.0±9.2
KNN - Edit	23.3±0.05	14±0.16	25.6±0.02	12±2.03
KNN - LCS	29.0±17.0	10.0±11.83	N/A	N/A
$1 Recognizer	**99.9±0.3**	**99.3±1.27**	95.1±1.64	85.5±5.52
HMM	76.8±4.53	78.1±6.47	72.6±3.88	58.6±7.98
LSTM	98.1±1.04	96.5±4.61	96.5±2.42	**93.7±8.26**
FFNN	97.7±1.55	95.3±2.49	**96.8±2.71**	89.4±6.7
LogisticRegression	98.5±1.28	94.7±4.34	96.7±2.57	88.8±7.73

The results, as shown in Table I, indicate that advanced models generally outperform baseline models across both domains, with the difference being more significant in the User-Independent (U-indep.) validation setting. In this scenario, models are evaluated on users not seen during training, which makes the task more challenging due to the variability in individual gesture patterns. Despite this, some models still achieve remarkable performance. This is particularly evident in

Domain 1, where models like the $1 Recognizer reach near-perfect accuracies above 99%, even without access to user-specific training data.

However, when applied to Domain 4, a more complex scenario, the performance of some models, especially those designed for 2D data (such as the $1 Recognizer), drops significantly. For Domain 4, the Feedforward Neural Network (FFNN) and Long Short-Term Memory (LSTM) models perform the best, with the LSTM model achieving the highest user-independent accuracy of 93.7%, demonstrating the model's ability to generalize better to 3D data compared to others.

The KNN-Edit and KNN-LCS models show notably poor performance across all settings, which can be attributed to their high sensitivity to hyperparameters, particularly the number of centroids used in the k-means clustering algorithm for labeling the data. Despite extensive tuning, these models remain inefficient, with KNN-Edit taking an average of 41 hours per run on our pipeline, making fine-tuning impractical.

Surprisingly, the combination of DTW with KNN, although considered a baseline model, performs exceptionally well in the User-Dependent setting, achieving very high accuracy. This indicates a strong ability to generalize within individual users. However, a major drawback becomes evident when the model is evaluated in the User-Independent setting, where performance drops significantly. This highlights its limitations for real-world applications involving unseen users during training.

VI. Discussion

In this section, we analyze and interpret the key findings from our experiments. We examine which models perform best under different conditions, how user behavior impacts recognition accuracy, and which gestures are more prone to misclassification. We also explore confusion patterns between gestures and reflect on the computational costs associated with certain models. These insights help clarify the strengths and limitations of the evaluated approaches and guide future work.

A. Best Performing Model on Combined Datasets

In this experiment, we combine the 10 subjects, including hand-drawn digits and shapes, across all domains. The goal is to evaluate and compare the performance of advanced models under both user-dependant (U-dep.) and user-independent (U-indep.) validation settings.

TABLE II: Comparison of model accuracies (%) under both validation settings for all Domains together. Higher is better.

Method	All Domain	
	U-dep.	U-indep.
$1 Recognizer	96.6±0.94	89.85±3.7
FFNN	**97.4±1.39**	91.75±4.21
LSTM	96.4±3.33	**94.7±4.98**
LogisticRegression	96.5±2.09	91.0±4.54

As shown in Table II, the Feedforward Neural Network (FFNN) achieves the highest accuracy of 97.4% in the User-Dependent (U-dep.) setting, demonstrating strong performance when user-specific patterns are present in the training data. However, in the more challenging User-Independent (U-indep.) setting, the Long Short-Term Memory (LSTM) model clearly outperforms the others, achieving an accuracy of 94.7%. This result highlights the LSTM's superior ability to capture temporal and sequential dependencies that generalize well across different users, making it particularly well-suited for real-world scenarios where the model must adapt to unseen users.

B. User performances

In this experiment, we assess the classification performance of each individual user to identify variations in gesture quality and consistency. Specifically, we aim to determine which users demonstrate the most stable and distinctive gestures, and which users struggle with producing recognizable inputs.

TABLE III: Per-user average accuracies across all advanced models for Domain 1, Domain 4, and combined. Higher is better.

User	Domain 1	Domain 4	Combined
0	90.75%	84.25%	87.50%
1	92.75%	89.00%	90.88%
2	94.75%	90.25%	92.50%
3	92.00%	91.50%	91.75%
4	**95.50%**	89.50%	92.50%
5	94.00%	**93.75%**	**93.88%**
6	91.50%	92.25%	91.88%
7	**95.50%**	90.75%	93.12%
8	92.00%	89.00%	90.50%
9	91.50%	91.00%	91.25%

As shown in Table III, User 0 consistently yields the lowest classification accuracy across domains, indicating a lower drawing precision or less distinctive gesture patterns. In contrast, Users 5 and 7 achieve the highest overall scores, suggesting strong gesture clarity and consistency. Notably, User 9 displays balanced performance across both domains, indicating a stable drawing style that is resilient to domain complexity and variation.

C. Most and Least Consistent Gestures/Digits in Classification

In this experiment, we identify the gestures or digits that are most prone to misclassification and those consistently classified with high accuracy. The goal is to understand the factors behind poor performance and highlight the labels with the highest classification success.

TABLE IV: Misclassification percentage per gesture on Domain 1 and 4. This is an average over all advanced models. Lower is better.

Domain 1	Misclass. %	Domain 4	Misclass. %
Digit 0	**14.75%**	Cone	16.25%
Digit 1	13.25%	Cuboid	14.25%
Digit 2	12.75%	Cylinder	18.75%
Digit 3	6.75%	CylindricalPipe	30.00%
Digit 4	2.75%	Hemisphere	14.00%
Digit 5	11.75%	Pyramid	20.00%
Digit 6	7.75%	RectangularPipe	15.75%
Digit 7	4.75%	Sphere	18.50%
Digit 8	1.75%	**Tetrahedron**	**32.50%**
Digit 9	5.25%	Toroid	14.25%

As shown in Table IV, the most difficult labels to classify were *Digit 0* in Domain 1 and *Tetrahedron* in Domain 4, with misclassification rates of 14.75% and 32.50%, respectively. These high error rates are likely due to the visual or structural similarities these classes share with multiple other gestures, which can confuse the classifiers. For example, the *Tetrahedron* often overlaps in form with other polyhedral shapes, while *Digit 0* may resemble digits like 6 or 8 when hand-drawn with irregularity.

On the other hand, the most consistent gestures were *Digit 8* and Digit 4 in Domain 1, with the lowest misclassification rate of 1.75% and 2.75% respectively. These digits are visually distinct and less likely to be confused with other gestures, contributing to their high accuracy in classification. Similarly, in Domain 4, the *Cone* and *Cuboid* shapes had relatively low misclassification rates of 16.25% and 14.25%, respectively, indicating that these geometric shapes are easier to differentiate from others in the set.

D. Most and Least Confused Gestures/Digits in Classification

In this experiment, we analyze which gestures or digits contribute most significantly to classification confusion.

TABLE V: Confusion contribution per gesture on Domain 1 and 4 (as % of all off-diagonal confusions). This is an average over all advanced models. Lower is better.

Domain 1	Confusion %	Domain 4	Confusion %
Digit 0	9.20%	Cone	13.90%
Digit 1	16.56%	Cuboid	5.28%
Digit 2	**19.33%**	Cylinder	10.04%
Digit 3	7.98%	CylindricalPipe	12.61%
Digit 4	0.00%	Hemisphere	2.32%
Digit 5	10.43%	**Pyramid**	**16.34%**
Digit 6	12.88%	RectangularPipe	8.75%
Digit 7	3.37%	Sphere	11.07%
Digit 8	13.19%	Tetrahedron	11.07%
Digit 9	7.06%	Toroid	8.62%

As seen in Table V, the most frequently confused gestures in Domain 1 were *Digit 2* (19.33%) and *Digit 1* (16.56%), which are often confused with each other due to their similar shape. In Domain 4, *Pyramid* and *Tetrahedron* (16.34% and 13.90%, respectively) were the most confused shapes, likely because of their similar structural features.

Conversely, the least confused gestures were *Digit 4* (0.00%) and *Hemisphere* (2.32%), which are visually distinct from others, leading to a low confusion rate. These gestures are more easily classified correctly due to their unique visual features.

The most frequently confused pairs in Domain 1 were:

- Digit 1 $\leftrightarrow$ Digit 2 (11.96%)
- Digit 5 $\leftrightarrow$ Digit 6 (5.21%)
- Digit 0 $\leftrightarrow$ Digit 6 (3.68%)
- Digit 0 $\leftrightarrow$ Digit 8 (3.68%)
- Digit 2 $\leftrightarrow$ Digit 3 (2.61%)

For Domain 4, the most frequently confused pairs were:

- *Pyramid* $\leftrightarrow$ *Tetrahedron* (8.49%)
- *CylindricalPipe* $\leftrightarrow$ *RectangularPipe* (3.60%)
- *CylindricalPipe* $\leftrightarrow$ *Sphere* (3.60%)
- *Cone* $\leftrightarrow$ *Tetrahedron* (3.47%)
- *Cuboid* $\leftrightarrow$ *Cylinder* (2.12%)

It is worth noting that there is a strong correlation with the results of the previous section. The gestures that are most frequently confused are typically also those with the highest misclassification rates, which is logically consistent.

E. Computational Cost of Models

Almost every model in our pipeline operates efficiently and can be computed very quickly. However, there are three exceptions: DTW (Dynamic Time Warping), Edit Distance, and Longest Common Subsequence (LCS). The computational complexity of these distances is quadratic, which explains the significant performance bottleneck when calculating them, especially with larger datasets or longer sequences. This is why our pipeline struggles to process these distances within a reasonable time frame.

However, several studies have explored approximation methods for these distances, which can drastically improve performance, particularly in scenarios where constant-time computations are crucial [3]. Future work could focus on implementing such approximation techniques to enable more efficient k-means fine-tuning, ultimately improving the computation of optimal solutions, particularly in offline scenarios.

VII. Conclusion

This study investigates the effectiveness of various machine learning techniques for classifying 3D hand-drawn gestures across two domains: digits (Domain 1) and geometric shapes (Domain 4). Our experiments demonstrate that advanced models generally outperform baseline approaches, with the LSTM model achieving superior generalization (94.7% accuracy) in user-independent scenarios, while the FFNN model excels (97.4% accuracy) when user-specific data is available. The $1 Recognizer, despite being designed for 2D recognition, performed remarkably well for digits (99.3% accuracy) but showed limitations with more complex 3D shapes. Our analysis reveals significant variations in both user performance and gesture difficulty. Certain gestures consistently presented classification challenges (Digit 0: 14.75% misclassification; Tetrahedron: 32.50%), with confusion patterns often following visual or structural similarities between gestures (Digit 1 $\leftrightarrow$ 2: 11.96%; Pyramid $\leftrightarrow$ Tetrahedron: 8.49%). From a computational perspective, while some distance-based methods like DTW offered competitive performance, approaches such as Edit Distance proved prohibitively expensive, highlighting practical trade-offs between accuracy and efficiency. For future work, exploring approximation techniques for computationally intensive distance metrics could enable more efficient implementations while maintaining classification accuracy. Additionally, investigating hybrid approaches that leverage the complementary strengths of different models and developing user-adaptive techniques could further enhance performance across diverse application scenarios. As spatial computing and gestural interfaces continue to evolve, the insights from this

comparative study provide valuable benchmarks for developing more intuitive and responsive human-computer interaction systems.

REFERENCES

[1] R. Huang, A. Jaiswal, and R. Rai. *Gesture-based system for next generation natural and intuitive interfaces*. In *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 33(1):54–68, 2019.

[2] Wobbrock, J.O., Wilson, A.D., and Li, Y. (2007). *Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes*.

[3] Mahdi Boroujeni, Masoud Seddighin, and Saeed Seddighin. *Improved algorithms for edit distance and LCS: beyond worst case*. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '20)*, pages 1601–1620, Society for Industrial and Applied Mathematics, 2020.

[4] L. Bergroth, H. Hakonen, and T. Raita. *A survey of longest common subsequence algorithms*. In *Proceedings of the Seventh International Symposium on String Processing and Information Retrieval (SPIRE 2000)*, pages 39–48, A Coruna, Spain, September 7–29, 2000. IEEE Computer Society.