Spring 5-24-2021

American Sign Language Assistant

Charulata Lodha

American Sign Language Assistant

A Project

Presented to

The Faculty of Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Charulata Lodha

Spring 2021

The Designated Project Committee

Approves

the Master's Project Titled

American Sign Language Assistant

By

Charulata Lodha

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

Spring 2021

Dr. Christopher Pollett	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Ms. Shruti Kothari	Amazon

ABSTRACT

American Sign Language Assistant

By

Charulata Lodha

Our implementation of a prototype computer vision system to help the deaf and mute communicate in a shopping setting. Our system uses live video feeds to recognize American Sign Language (ASL) gestures and notify shop clerks of deaf and mute patrons' intents. It generates a video dataset in the Unity Game Engine of 3D humanoid models in a shop setting performing ASL signs.

Our system uses OpenPose to detect and recognize the bone points of the human body from the live feed. The system then represents the motion sequences as high dimensional skeleton joint point trajectories followed by a time-warping technique to generate a temporal RGB image using the Seq2Im technique. This image is then fed to the image classification algorithms that classify the gesture performed to the shop clerk.

We carried out experiments to analyze the performance of this methodology on the Leap Motion Controller dataset and NTU RGB+D dataset using the SVM and LeNet-5 models. We also tested 3D vs 2D bone point dataset performance and found 90% accuracy for the 2D skeleton dataset.

Keywords – Unity, Convolutional Neural Network (CNN), American Sign Language(ASL)

ACKNOWLEDGEMENTS

This project is a result of constant support and guidance from Dr. Christopher Pollett. I would like to express my sincerest gratitude to him for his unwavering supervision, patience, and motivation throughout my project. I consider myself to be extremely fortunate to have had an opportunity to work with someone as learned and humble as him.

I am also grateful to my committee members, Dr. Robert Chun and Shruti Kothari for providing their valuable feedback and guidance. I would also like to thank the CS department for providing a thriving environment for success.

And finally, I thank my supportive parents and friends for always having my back and helping me work on something so meaningful.

LIST OF FIGURES AND TABLES

Figure 1: ASL Alphabets 6
Figure 2: Red Sign in ASL 6
Figure 3: Pay Sign in ASL 7
Figure 4: LeNet-5 Architecture 8
Figure 5: SVM Classifier 9
Figure 6: SVM Classifier - Outlier 9
Figure 7: SVM Non-Linear Hyperplane Example 10
Figure 8: Parameters of SVM 10
Figure 9: Add Property in Animation 12
Figure 10: Animation configuration of Okay Sign in Dopesheet mode 12
Figure 11: Animation configuration of Okay Sign in Curve Mode 13
Figure 12: Okay Sign Unity Animation Clip Snapshots 13
Figure 13: Unity Recorder 14
Figure 14: Input Video to OpenPose 16
Figure 15: Output Video of OpenPose 15
Figure 16: XYZ Mapping to RGB Space 16
Figure 17: Sample output temporal image of human jump 16
Figure 18: Application Design 17
Figure 19: Project Architecture 19
Figure 20: MNIST ASL Dataset Preview 20
Figure 21: NTU RGB Action Classes 21
Figure 22: NTU RGB Skeleton Points 22
Figure 23: Hand Joint Points as seen by Leap Motion Controller 23
Figure 24: LMC Orange Left Hand Dataset Preview 23
Figure 25: LMC Orange Right Hand Dataset Preview 24
Figure 26: C# code Snippet 1 25
Figure 27: C# code Snippet 2 25
Figure 28: LeNet-5 Model Architecture 26
Figure 29: LeNet-5 ASL Alphabet Detection Result : Y class 26
Figure 30: LeNet-5 Keras Model for ASL 27
Figure 31: SVM Model for ASL 28
Figure 32: MNIST ASL Alphabets on LeNet-5 model Epoch Accuracies 30
Figure 33: Training Accuracy for LeNet-5 ASL Alphabet Detection 31
Figure 34: NTU RGB Classification Report for SVM 32
Figure 35: NTU RGB Training Accuracy and Loss Graph 32
Figure 36: SVM Classification Report for NTU RGB Dataset 33

Figure 37: Classification Report for SVM for 9 classes of LMC 34
Figure 38: Output Skeleton visualized from OpenPose 36
Figure 39: Skeleton coordinates generated from OpenPose 36
Figure 40: 3D v/s 2D temporal mapping for Class 2 37
Figure 41: 3D v/s 2D temporal mapping for Class 23 37
Figure 42: 3D v/s 2D temporal mapping for Class 38 37
Figure 43: SVM Classification Report for 2D NTURGB Dataset 38

Table 1: Sign Categories based on Depth 7
Table 2: Action Response Table 18

Table of Contents

I. Introduction 1

II. Background 5

a. Shopping Center ASL 5

b. LeNet – 5 Architecture 7

c. Support Vector Machine 8

d. Unity Game Engine 11

e. OpenPose Model 14

f. Temporal Representation of 3D Skeletal Action Sequences 15

III. Design 17

a. Application Design 17

b. Project Architecture 18

IV. Implementation 20

a. Dataset 20

b. Unity Animation Creation and Scripting 24

c. ASL Alphabet Recognition in Real-Time 25

d. ASL Word Recognition in Real-Time 27

V. Experiments 29

a. MNIST ASL Alphabet Dataset 29

b. NTU-RGB v/s LMC for SVM & LeNet-5 model 31

c. Okay Sign from Unity Dataset 35

d. Impact on accuracy 2d vs 3d co-ordinates 36

VI. Conclusion 39

References 40

I. INTRODUCTION

In the United States between 6 and 8 million people suffer from language impairment [1]. Sign language focuses mainly on hands gestures for communication. In some cases, the body, head, and facial expression are also accounted to accurately understand a sign's intended meaning. This report describes an approach for building a digital assistance system that provides a solution to in-shop issues for mute and deaf people by providing live communication between shop keeper and customers through video feeds using state-of-the-art Convolution Neural Networks and Unity Game Engine.

American Sign Language (ASL) is one of the most widely used sign language in the U.S. [2]. ASL is not sign language English but a visual language where signs convey ideas rather than literal words. As a result, there comes an undeniable communication barrier between the ASL and English speaking population.

The barrier in communication between people obstructs the normal way of living. For example, if a deaf person at the shopping mall wants to seek some suggestions for clothing then and if there no ASL-speaking people around then one might not get the same seamless shopping experience just like the one with the ability to speak English. Whether one is a store owner or government, it's high time that we enforce a mechanism to maintain stores that are accessible for deaf customers.

The Americans with Disabilities Act (ADA) prohibits any kind of disability-based discrimination[3]. It is also responsible for enforcement of the law to ensure that people with disabilities can access any online content [3]. As the world is tremendously progressing in accessibility technologies, a resilient computer system that provides real-time communication

must be built and mandated in public places to aid those hard of hearing or deaf.

The existing accessibility technologies show an innovative application of Augmented Reality, Machine Learning, Artificial Intelligence, and Human-Computer Interaction to better accommodate mute and deaf people. HoloHear has used HoloLens and prototyped an augmented reality app in which as a customer speaks aloud, a 3D holographic model appears that translates ASL to English in real-time [4]. TapSOS app allows a person who is mute and deaf to seek help from emergency services without having to speak to it by just tapping in the app to a particular issue like breathing issue, fire and others to inform about the respective authorities and won the 2018 Digital Health Award [5]. Today shops provide amplifiers to deaf people to adequately communicate with the sales assistant [6].

Convolutional Neural Networks have proven to be extremely successful for image recognition and classification problems. It has been widely implemented for recognizing human gestures lately. In [7] J. J. Bird, et al show that a late fusion approach to multimodality in sign language recognition improves the overall ability of the model in comparison to the singular approaches of image classification (88.14%) and Leap Motion data classification (72.73%).

Another approach to perform gesture recognition is through human temporal dynamics. The Skeleton-based human action recognition is a time-series problem used by [8]. The skeleton dataset comprises the human key joints in 3D space over time. This approach is possibly a more detailed representation of gesture than just a few snapshots over time as here each action is being accounted by motion of skeleton sequences as a series of time frames.

The rise of computing technologies motivated the development of Kinect and Leap Motion that are used as an input device to capture motions. These human-machine interaction device finds its application in many fields from augmented reality to healthcare.

The availability of easy-to-use depth sensors like Microsoft Kinect™ has promoted research work in the computer vision community specifically in action recognition by providing 3D skeletal characteristic of human body movements. This skeleton information is quite complex for traditional 2D cameras[9]. This further alleviates the problem of hand gesture recognition by providing the CNNs with temporal information about each of the body joint points and gives us the motivation for this project.

The primary contribution of this project is to present an accessibility CNN-based solution that is an economically viable and easily integrable end-to-end system for a shopping center to recognize the intent of the deaf and mute that would help them independently shop and have a seamless shopping experience. It simplifies the process of gesture recognition by transforming the skeleton sequences into RGB temporal dynamics which is fed as input to CNN-based image recognition models and thus highly reducing the training time and complexity. It also gives the owner the ability to create their dataset as CNNs require a huge amount of training data to avoid overfitting. This project presents a way for a generation of a great quality dataset using Unity animations and C# scripting that could be potentially be used for computer vision research work. This acts as an alternative to having an infrastructure with huge resources, manpower as well as the cost to generate quality datasets. It also draws insights into how the training size of the dataset impacts the accuracy of the experiments performed on various datasets. This will help the user make critical decisions concerning the generation of their dataset from Unity and partitioning it while training the model for the best accuracy.

The project report is organized into chapters as follows: Chapter 2 defines the background and concepts used to build the final system, why we used LeNet-5 architecture, prediction of signed alphabets, animation creation in Unity Game Engine as well scripting

required for the generation of gesture dataset from Unity. Lastly, it shows how to leverage OpenPose for the detection of body joint points and conversion of gesture frames into RGB temporal images. Chapter 3 talks in detail about the design of the project overall. Chapter 4 describes the end-to-end implementation of our project and the setup we used to conduct the experiments. Chapter 5 describes those experiments that we conducted and gives their results. Finally, in Chapter 6 we have our conclusion.

II. BACKGROUND

Our purpose is to design an application leveraging state-of-the-art neural networks to classify and predict the ASL gestures in real-time. In this section, we will briefly review works related to dataset generation and building our final model. This section outlines the background on technical details about ASL Alphabets and most widely used words in the shopping center, LeNet-5 architecture, Unity Animations, OpenPose to understand how to generate body key points, and lastly about how to convert Motion Capture Sequences as 2D-RGB images from Skeleton dataset.

a. SHOPPING CENTER ASL

Learning about ASL signs and analyzing them with the perspective of building an AI model is focused on in this sub-section. The ASL alphabets have 24 static and 2 motion hand gestures for J and Z namely as Figure 1. A model was built to recognize the static alphabets that let the customer sign their name in ASL and convey it to the shop clerk. When a person shops, the most basic and widely used words to communicate are “Hello”, “Cost”, “Offer”, “Color”, “Size”, “Pay”, “Okay”, “No”, and “Thank You”. While all of these signs involve motion, some of them are more complex and to build a robust model more data needs to be gathered rather than just static images of the sign.

Figure 1: ASL Alphabets

If we consider a sign in 3 dimensions, then X, Y, and Z-axis provide different views of the sign in various planer regions. If we are viewing it from the X-Y plane, then to understand the peculiarities of a gesture better, depth information is required. This is obtained by getting the depth of each key point in the Z direction.

The sign for "red" is performed by using just the index finger. It is done as if one is stroking lips with an index finger's tip. This gesture doesn't require much depth information as it doesn't involve any motion backward i.e. towards the body.

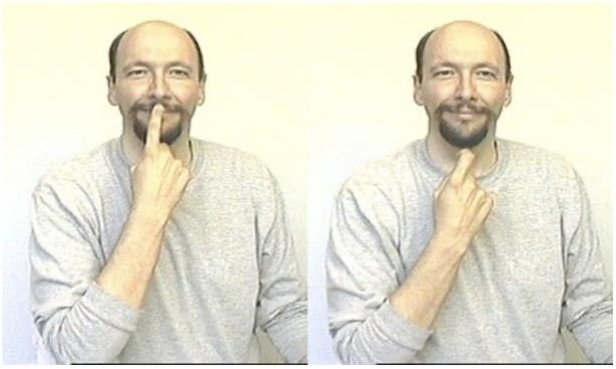


Figure 2: Red Sign in ASL

Now, if we consider the sign for "pay", then it has a gesture in the Z direction using the

index finger. So, here depth information is most crucial to analyzing the gestures.

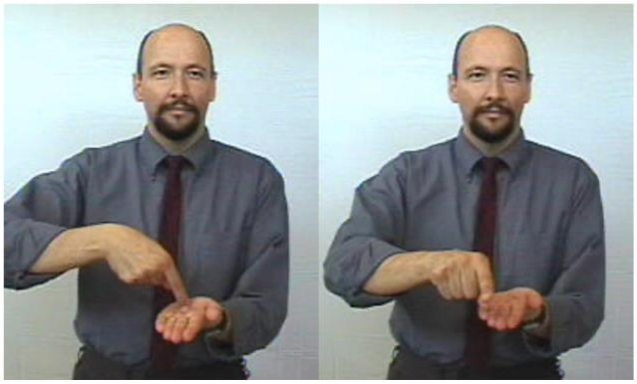


Figure 3: Pay Sign in ASL

Based on this criterion of depth, signs are classified into 2D and 3D signs as listed in Table 1.

Sign Category	**Signs**
2 D Motion ASL Gesture	Red, Yellow, Hand Wave, Okay,
3 D Motion ASL Gesture	Blue, Pay, Hello, Eat, Thank You, Offer

Table 1: Sign Categories based on Depth

b. LeNet – 5 Architecture

Convolution Neural Networks are the standard form of multi-layer neural networks that helps in solving recognition problems related to images. Specific architectures of CNN are designed for solving problems like object detection, pose estimation, and others. CNN is capable of recognizing visual patterns in an image through pixel processing. Our project aims at finding visual patterns in the temporal representation of ASL action sequences. In this subsection, we will explore more LeNet-5 as will be using it as our base architecture for all the implementations.

The LeNet-5 architecture is one of the earliest to be used for deep learning and employs a backtracking algorithm. Yann LeCun showed that minimizing the number of free parameters in

neural networks can enhance the generalization ability of neural networks. This architecture was used for the recognition of the handwritten and machine-printed characters. It is widely used for various image recognition problems because of its simple and quite elementary architecture.

It has a total of two convolutional layer, two pooling layers, followed by a flattening convolutional layer, then two fully connected layers, and lastly a SoftMax classifier [10] as shown in Figure 2. The number of training parameters is approximately seventy thousand.

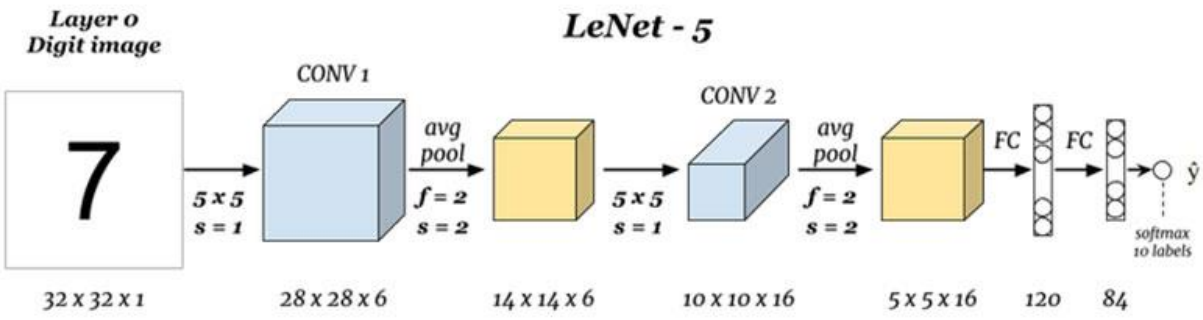


Figure 4: LeNet-5 Architecture

c. SUPPORT VECTOR MACHINE

SVM (Support Vector Machine) is a machine learning model and falls into the category of supervised learning. It has been widely used for classification and regression analysis as SVMs are proved to one of the most robust prediction methods based on statistical learning frameworks [11]. SVM is capable of producing results on complex datasets that are smaller in size. Our datasets are temporal images that are very complex for the general object recognition model. So, for this project will run our experiments for various datasets on SVM as well with LeNet-5 to analyze the accuracy of our model. Hence, understanding the working of the SVM

model is important.

In the SVM algorithm, we start by plotting every single data as a point in n-dimensional space where n is defined as the number of features. Each coordinate represents the value of each feature. Next, we start the process of segregating the data points into certain categories. For this, a hyperplane is searched that separates each category distinctly as shown in Figure 5.

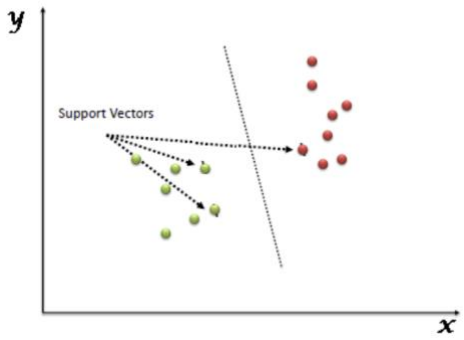


Figure 5: SVM Classifier

To find the right segregation hyperplane, among the multiple possibilities can be done by finding the Margin which is the distance between the nearest data point to the hyperplane and the hyperplane. If a low margin data plane is selected, then it might result in miss classification.

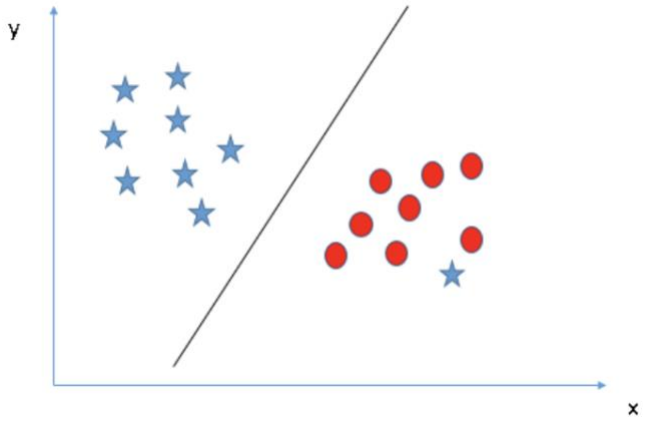


Figure 6: SVM Classifier - Outlier

SVM can also be used to solve problems where linear hyperplane is not possible for

example as shown in Figure 5. In this case, an additional feature is added to solve it which results in the equation z=x^2 + y^2 [12].

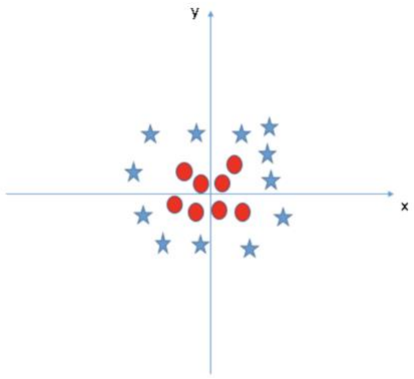


Figure 7: SVM Non-Linear Hyperplane Example

```
param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
 ]
svc = svm.SVC()
clf = GridSearchCV(svc, param_grid)
clf.fit(X_train, y_train)
```

Figure 8: Parameters of SVM

SVM can be implemented using sci-kit-learn library in Python. There are various parameters like kernel, gamma, and C that need to be tuned to get the best accuracy and avoid overfitting of the model as seen in Figure 8. To control the error, C is used as the penalty parameter. It is used to do a tradeoff between the classification of the training points correctly and smooth decision boundaries. Gamma is the kernel coefficient for sigmoid, rbf, and poly. A lower value of gamma will lead the training data into efficient segregation [12]. But higher value may cause generalization error and the issue of overfitting of the data. SVM can ignore outliers. It can determine the maximum margin required to find the hyper-plane. Therefore, SVM

classifications are said to be robust to outliers and hence prove efficient over other machine learning models.

d. UNITY GAME ENGINE

For any computer vision model, the efficiency relies on the quality and size of the dataset. Many times, we might not have the exact dataset we need. So, having an approach that is easier to implement can certainly solve this problem. In this subsection, we present a way of leveraging animations on a humanoid avatar in Unity Game Engine. It outlines how to make an avatar that does a sign language gesture in ASL. It then uses a script that places a camera in random locations to generate a varied video dataset that captures gestures done by a humanoid avatar from different angles in the 3-dimensional space of the Unity Scene.

In Unity, we explored Animations, Skeleton Rigs configuration that helps in marking the bones points, Dopesheet that helps in configuring motion of avatar at a specific interval of time during the clip, Workflow creation in Animator that helps in designing the overall flow from the start state to exit state having transitions between multiple animations, light setting that helps in setting the Sun direction and Camera light, shopping center related custom object placement and finally learned how to add properties in Animations that helps in making the avatar do a custom motion for that specific body part as shown in Fig 9. A combination of animators is used to make an avatar perform specific gestures.

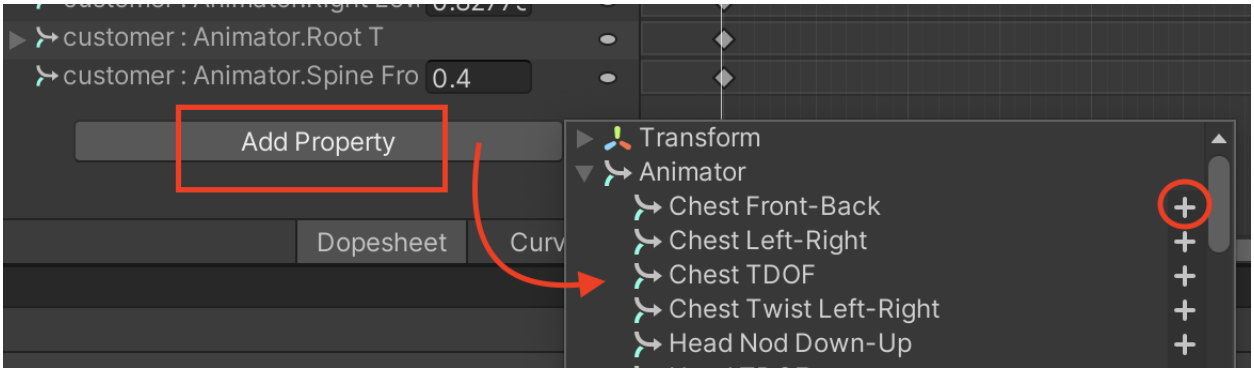


Figure 9: Add Property in Animation

The Animation timeline view has two modes, Dopesheets, and Curves. In Unity, Dopesheets allows viewing each property's keyframe as shown in Fig 10. The Animation Curve gives control of a property at a particular instant of time shown in Fig 10. The final result is Unity Scene in a shopping setting where the customer is making an Okay sign gesture to communicate to the shop owner as shown in Fig 11.

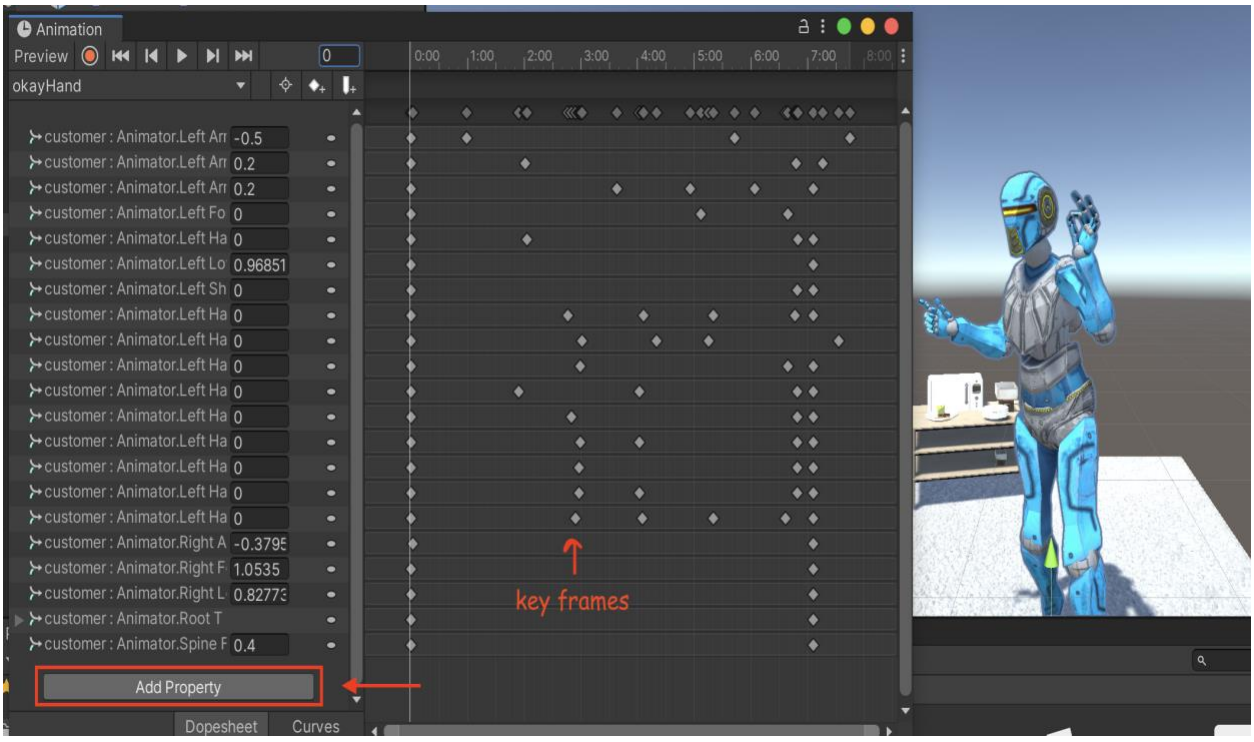


Figure 10: Animation configuration of Okay Sign in Dopesheet mode

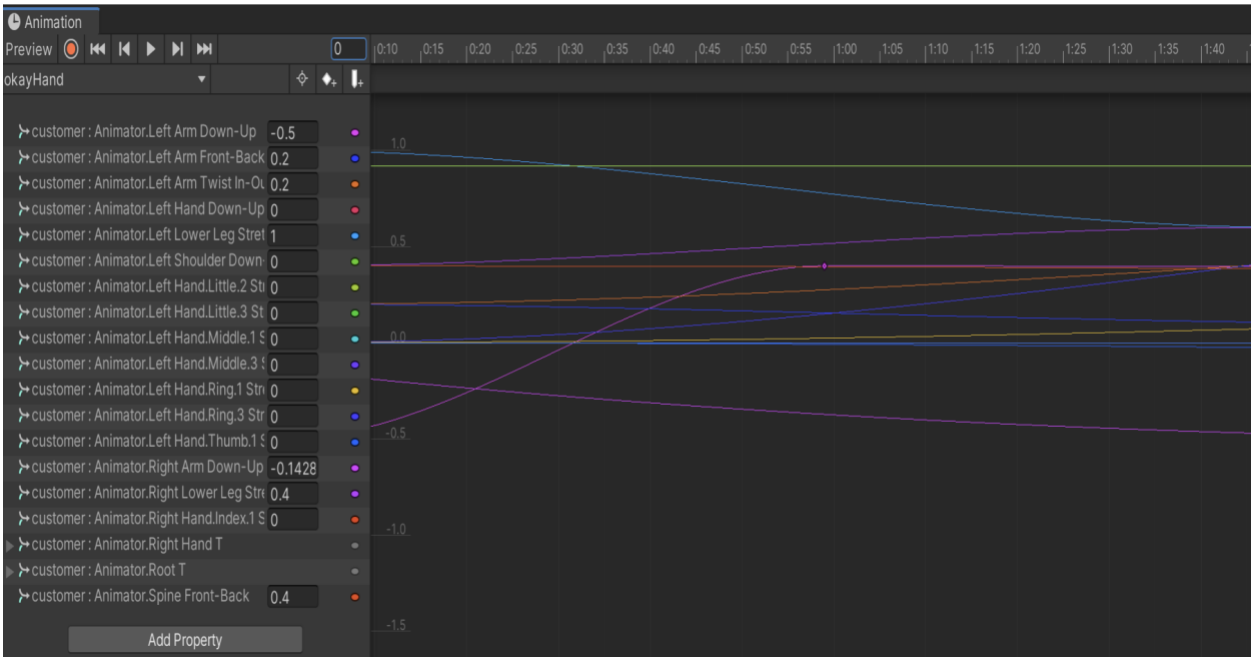


Figure 11: Animation configuration of Okay Sign in Curve Mode

Figure 12: Okay Sign Unity Animation Clip Snapshots

After the animation is ready, Unity Recorder is used to record the video in mp4 format as

shown in Fig 12. To get a quality dataset and maintain heterogeneity in each video, each of the videos in the dataset needs to be created by recording it from various angles in the Unity scene. Now to generate these large number of videos, manually configuring the camera each time with different angles and appropriately placement of it in 3D space in the Unity Scene is not a feasible approach. So, we used a C# script to achieve the said requirement and attached it to the camera object.

In Unity, the C# scripts are usually created within it directly that can be attached to GameObjects [13]. So, here we attached this script to the main camera in Unity.

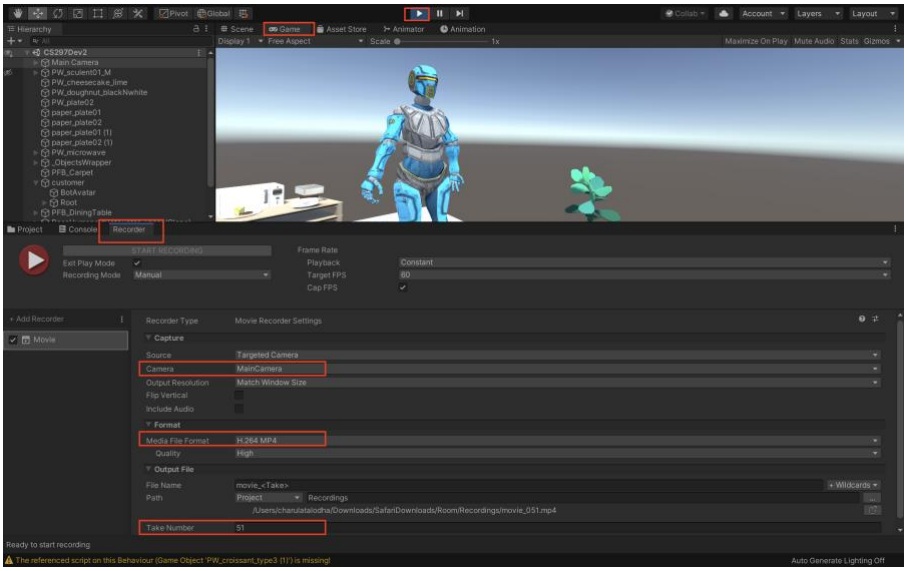


Figure 13: Unity Recorder

e. OpenPose Model

OpenPose is a real-time approach for multi-person key point detection: body, foot, hand, and facial key points [14]. For this project, we used 2D real-time multi-person key point detection, the output of which is being fed to get the 3D temporal mapping of skeleton frames.

The custom Unity videos as seen in Fig 13 are fed to the OpenPose model to generate a new dataset that has bone points marking in the video along with an output JSON file having 2D coordinates for key points of the human body as shown in Fig 14.

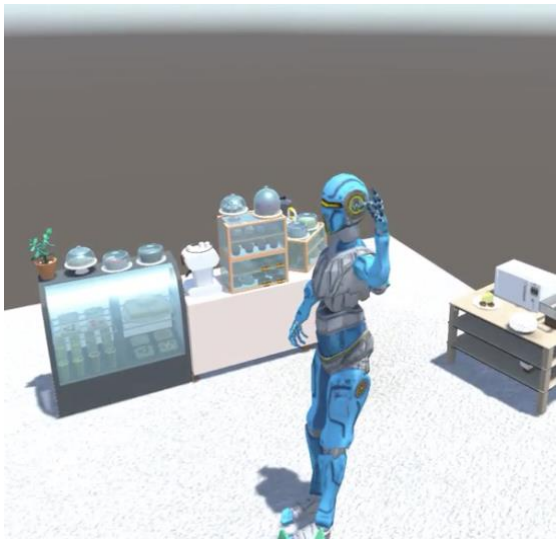


Figure 14: Input Video to OpenPose

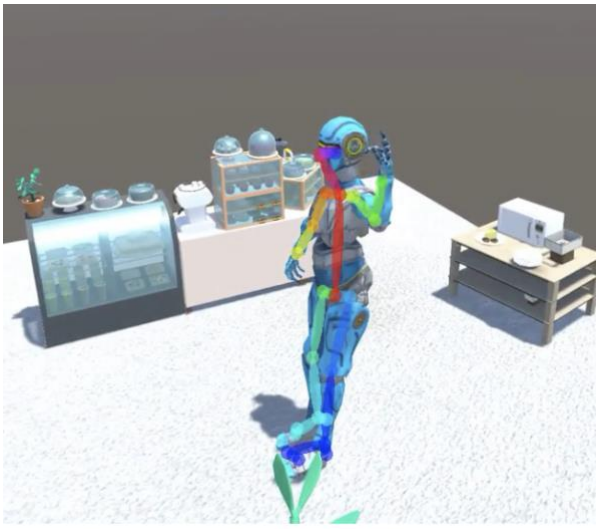


Figure 15: Output Video of OpenPose

f. Temporal Representation of 3D Skeletal Action Sequences

The motion sequences for the human skeleton can be represented as 3-dimensional trajectories [15][16][17]. In [18] Sohaib, et al proposed a way to transform a sequence into an RGB image. The 3D skeleton data is normalized and mapped into RGB space as shown in Fig. 16. This results in reducing the high dimensionality of motion capture sequences to 2D color images. For this project for all the motion-based ASL gestures, we employed this methodology and then performed image recognition using SVM and LeNet-5 models to classify and detect ASL gestures.

For example, if a human jump twice then it's a vertical direction motion over some time. When this is mapped to RGB space, 2 green patterns can be seen as shown in Fig 16. Also, as a

person jumps, all his body joint points have an upward motion with it i.e. in Y-Axis, and thus clear broad green bars can be seen in the image as Y maps to green color.

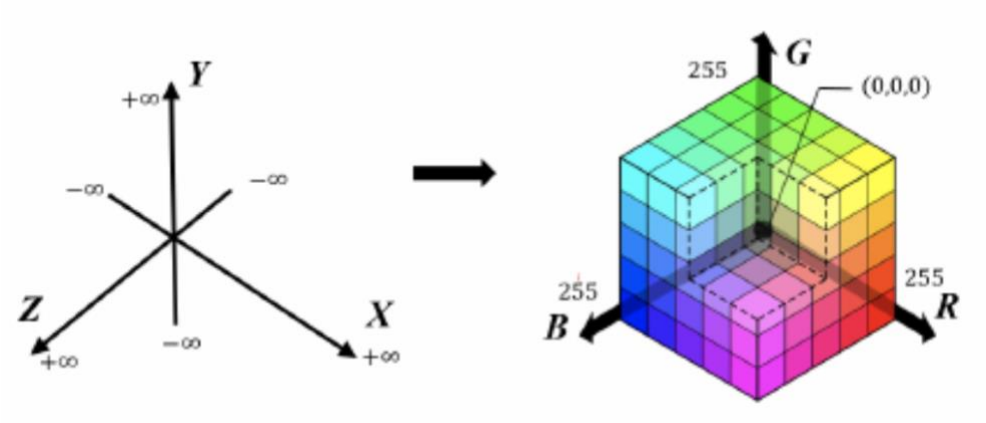


Figure 16: XYZ Mapping to RGB Space

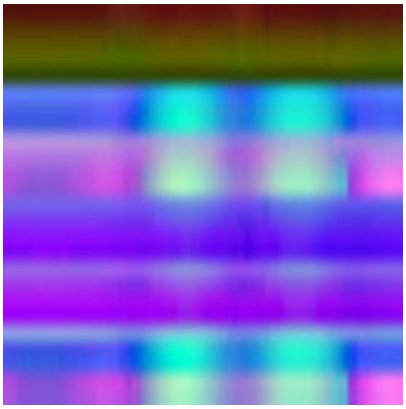


Figure 17: Sample output temporal image of human jump

III. DESIGN

a. APPLICATION DESIGN

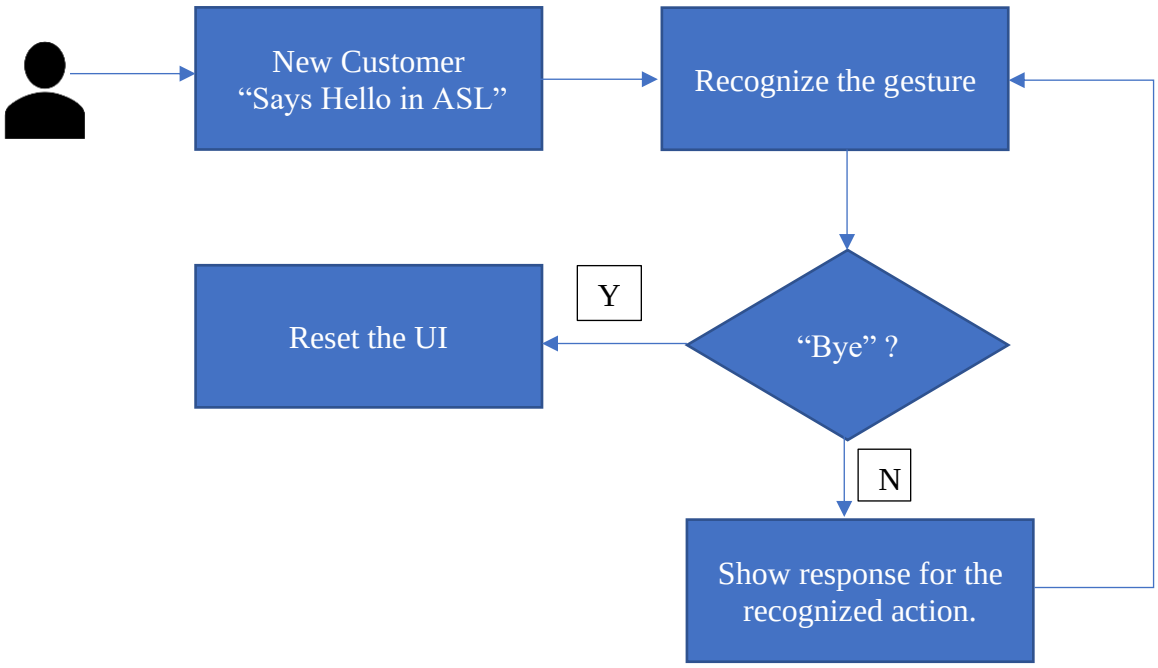


Figure 18: Application Design

As the application starts, the user activates the application by saying "hello" in ASL. When the gesture is recognized as "hello" correctly by the model it takes the user to the homepage screen. In that live video, feed is continuously captured and gestures are recognized. Once the ASL sign is recognized and classified, then the corresponding response will be given by the system. For example, we the customer does a calling gesture then the system will notify the clerk to come to visit the customer to provide support. Until the gesture is a "Bye" sign in ASL, a user is allowed to interact with the application.

Sample Use cases:

Action in ASL	**Response**
Mobile Sign	Call clerk for help
Brush Teeth	Show navigation map to reach the Toiletry section.
Make Victory Sign	Show the way to the washroom

Table 2: Action Response Table

b. PROJECT ARCHITECTURE

In this system, as soon as the user starts performing ASL gestures, the live feed is captured and passed to OpenPose. The OpenPose outputs frame sequences from the video feed representing the 27-body joint point skeleton sequences. Then human skeleton coordinates specifically focused on hands are recognized for each frame of the video. The first major step towards effective sign language recognition is capturing these critical components, but a much more complex process is required to turn this data into meaningful information. These time sequence skeletons are then transformed into an RGB image using the Temporal Image Generator and finally passed on to the image classifier as an input image. The classifier classifies the gesture and gives the predicted class label for the ASL word to the user as shown in Fig. 18.

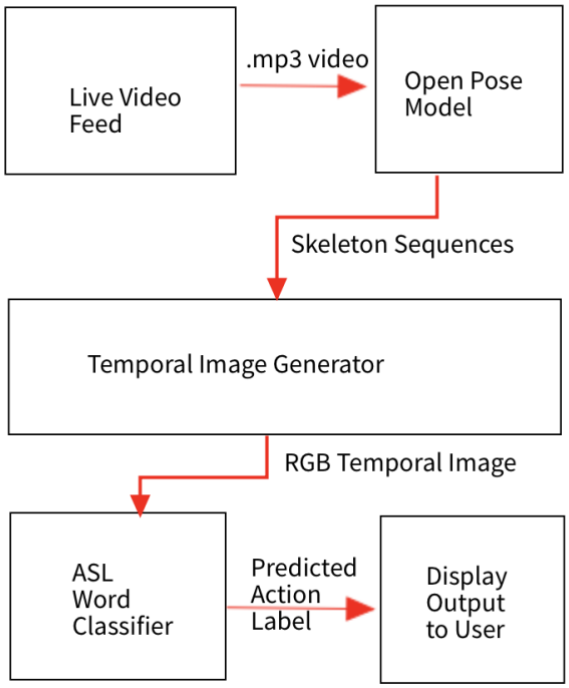


Figure 19: Project Architecture

IV. IMPLEMENTATION

This section focuses on datasets used, model architecture implemented, and methodology followed in this project. It also describes the configuration done for the image recognition model.

a. DATASET

1. MNIST ASL Alphabets Dataset

The MNIST ASL alphabets dataset [21] of hand gestures represent a multi-class problem with 24 classes of all letters except J and Z. Unlike all the letters that have static gestures, J and Z are motion-based gestures hence excluded from this dataset.

The dataset format draws its inspiration MNIST. Each training and test case represent a label from 0 to 25 map to A-Z. For motion gestures J=9 or Z=25 because of gesture motions. The training data has 27,455 and test data has 7,172. This dataset is provided as a .csv file that has 1 column for label and 784 columns for pixel values of the gesture image. This dataset thus helps in identifying the letter signed by the customer in a shop setting when they want to sign their name.

Dataset Sample Preview:

	label	pixel1	pixel2	pixel3	...	pixel781	pixel782	pixel783	pixel784
0	3	107	118	127	...	206	204	203	202
1	6	155	157	156	...	175	103	135	149
2	2	187	188	188	...	198	195	194	195
3	2	211	211	212	...	225	222	229	163
4	13	164	167	170	...	157	163	164	179
...	...	...	...	...	...	...	...	...	...
27450	13	189	189	190	...	234	200	222	225
27451	23	151	154	157	...	195	195	195	194
27452	18	174	174	174	...	203	202	200	200
27453	17	177	181	184	...	47	64	87	93
27454	23	179	180	180	...	197	205	209	215

[27455 rows x 785 columns]

Figure 20: MNIST ASL Dataset Preview

2. NTU-RGB+D Action Recognition Dataset

NTU-RGB+D dataset [19][20] contains 56,880 videos captured from Kinect V2 cameras providing RGB videos, 3D skeletal data, depth map sequences for each video sample. In all, it represents 60 action classes as shown in Fig. 21. There is another extended dataset called NTU RGB+D 120 has 120 classes and 114,480 videos. For this project, we used classes that represent actions that map to ASL gestures example eat is food in ASL, salute is hello in ASL, handwave is goodbye. We used these classes to analyze the efficiency of the approach presented in this project as this has been widely used as a benchmark in [18][19][20] for action recognition. The skeleton dataset used has 25 body joints co-ordinates as shown in Fig. 22 in a .skeleton file.

1.1 Daily Actions (82)

A1: drink water	A2: eat meal	A3: brush teeth	A4: brush hair
A5: drop	A6: pick up	A7: throw	A8: sit down
A9: stand up	A10: clapping	A11: reading	A12: writing
A13: tear up paper	A14: put on jacket	A15: take off jacket	A16: put on a shoe
A17: take off a shoe	A18: put on glasses	A19: take off glasses	A20: put on a hat/cap
A21: take off a hat/cap	A22: cheer up	A23: hand waving	A24: kicking something
A25: reach into pocket	A26: hopping	A27: jump up	A28: phone call
A29: play with phone/tablet	A30: type on a keyboard	A31: point to something	A32: taking a selfie
A33: check time (from watch)	A34: rub two hands	A35: nod head/bow	A36: shake head
A37: wipe face	A38: salute	A39: put palms together	A40: cross hands in front
A61: put on headphone	A62: take off headphone	A63: shoot at basket	A64: bounce ball
A65: tennis bat swing	A66: juggle table tennis ball	A67: hush	A68: flick hair
A69: thumb up	A70: thumb down	A71: make OK sign	A72: make victory sign
A73: staple book	A74: counting money	A75: cutting nails	A76: cutting paper
A77: snap fingers	A78: open bottle	A79: sniff/smell	A80: squat down
A81: toss a coin	A82: fold paper	A83: ball up paper	A84: play magic cube
A85: apply cream on face	A86: apply cream on hand	A87: put on bag	A88: take off bag
A89: put object into bag	A90: take object out of bag	A91: open a box	A92: move heavy objects
A93: shake fist	A94: throw up cap/hat	A95: capitulate	A96: cross arms
A97: arm circles	A98: arm swings	A99: run on the spot	A100: butt kicks
A101: cross toe touch	A102: side kick	-	-

Figure 21: NTU-RGB+D Action Classes

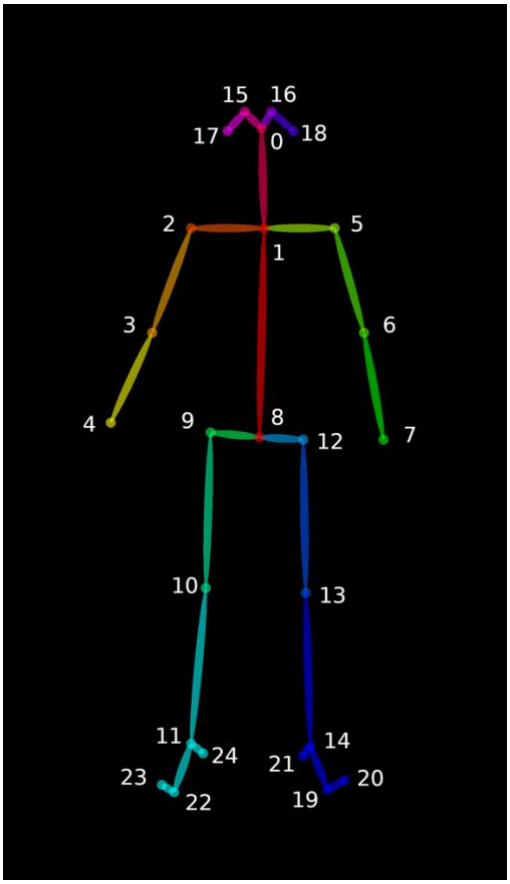


Figure 22: OpenPose Skeleton Points[14]

3. ASL Leap Motion Controller Dataset

This dataset [22] contains 25 subjects performing 60 different signs of the ASL and includes more than 17,000 signs in total. The dataset is composed of the joint positions provided by the Leap Motion API for both hands as shown in Fig. 23. The dataset contains ASL gesture classes like red, blue, yellow, come, cost, shop, big, small, and others that are widely used in a shop setting and hence are idle for our project.

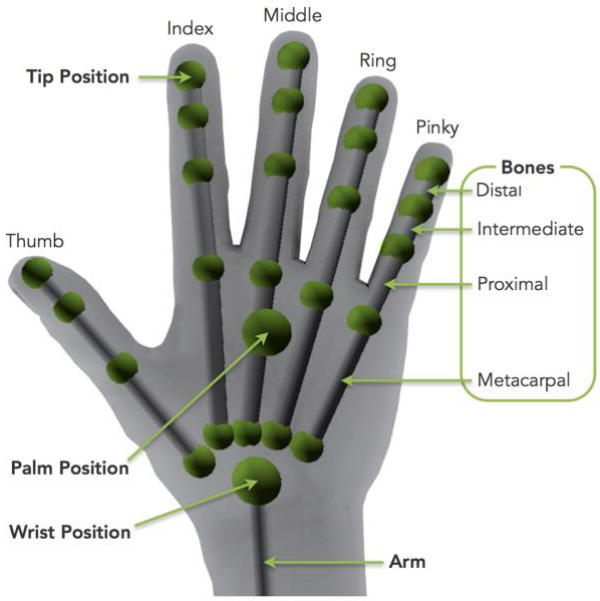


Figure 23: Hand Joint Points as seen by Leap Motion Controller[22]

Every action has a separate dataset file for the left and right hand. Each of these files represents the skeleton motion trajectory in 3D for that single hand. So, gestures that need both hands to sign focuses on combined results from left & right-hand datasets. For gestures that are performed by a single hand, then right-hand files contain the motion and the left-hand motion are generally constant and hence can be ignored as seen in Fig. 24 and Fig. 25.

	Time	thumbProximal_L_X	thumbProximal_L_Y	thumbProximal_L_Z	thumbDistal_L_X
0	21:34:36.368 PM	-0.1097075	0.01622243	0.39756579999999997	-0.11329960000000001
1	21:34:36.395 PM	-0.10988230000000002	0.01601733	0.3980123	-0.1135501
2	21:34:36.425 PM	-0.1100047	0.01598163	0.39820559999999994	-0.11370119999999999
3	21:34:36.456 PM	-0.1098692	0.01586817	0.39831500000000003	-0.11356589999999998
4	21:34:36.485 PM	-0.10983820000000001	0.01578118	0.3984414	-0.11358519999999998
5	21:34:36.517 PM	-0.10987960000000001	0.01576672	0.39838850000000003	-0.11361500000000001

Figure 24: LMC Orange Left Hand Dataset Preview

	Time	thumbProximal_L_X	thumbProximal_L_Y	thumbProximal_L_Z	thumbDistal_L_X
0	21:34:36.368 PM	0.11964489999999998	0.02544089	0.40602309999999997	0.11842480000000001
1	21:34:36.395 PM	0.1194022	0.02560966	0.40558299999999997	0.1180625
2	21:34:36.425 PM	0.1191847	0.02572824	0.40531690000000004	0.1177519
3	21:34:36.456 PM	0.1186636	0.02577735	0.40505399999999997	0.11715899999999999
4	21:34:36.485 PM	0.1180744	0.02620117	0.4046636	0.116481

Figure 25: LMC Orange Right Hand Dataset Preview

b. UNITY ANIMATION CREATION AND SCRIPTING

We start by creating a Unity Scene where all the objects related to Shopping Center like a side table, chairs, main counter, walls, and floor are added. Next humanoid avatars are placed in the 3D space. Then to perform rigs configuration for the animated skeletons of these humanoid avatars that help in controlling its motion in the 3D space using Unity Animations. We start by selecting the Game object and creating a new empty Animation Clip in Unity. Then on the right side of the Animation View, one can see the timeline for the current clip. The keyframes for each animated property appear in this timeline.

The C# script has a method Start() which starts executing on the hit of the Play button runs run the unity project. To get started with some initial random values for the camera position, I used Random.Range(-2.0f, 2.0f) function. This gives a value from a minimum of -2.0f to a maximum of 2.0f. So, for all 3 directions, the min and max values are chosen to set a boundary of where a camera could be placed as shown in Fig. 26. This is like confining camera motion in a 3D space.

```
void Start()
{
    newlocation.x = Random.Range(-2.0f, 2.0f);
    newlocation.y = Random.Range(2.0f, 3.0f);
    newlocation.z = Random.Range(-3.0f, 5.0f);
}
```

Figure 26: C# code Snippet 1

The LateUpdate() is called after all the update methods are done with processing. If a unity object movement is part of the scene, then the camera should wait and track the object that might have moved. So, after the camera's random position is chosen in start(), the LateUpdate() is used to transform and rotate the camera object as shown in Fig. 27.

```
private void LateUpdate()
{
    if (customer == null)
        return;

    if (viewObj)
         transform.LookAt(customer);
    else
         transform.rotation = customer.rotation;

    if (offsetPositionSpace!= Space.Self)
        transform.position = newlocation + customer.position;
   else
        transform.position = customer.TransformPoint(newlocation);

}
```

Figure 27: C# code Snippet 2

c. ASL ALPHABET RECOGNITION IN REAL-TIME

ASL letter recognition system is a neural network to recognize letters of the American Sign Language (ASL). This implementation provides feature to recognize ASL alphabets gestures from a live video feed. The ASL alphabets have 2 motion hand gestures for J and Z. The rest of the 24 alphabet hand gestures are static. In this system, only static hand gesture

recognition is done using convolution neural networks. The network that we have designed has a total of 54,367 parameters based on LeNet-5 architecture as seen in Fig. 28 and it classifies the captures gesture and gives out a corresponding English alphabet in real-time as seen in Fig. 29.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 25)                3025
_________________________________________________________________
dense_2 (Dense)              (None, 25)                650
=================================================================
Total params: 54,367
Trainable params: 54,367
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 28: LeNet-5 Model Architecture

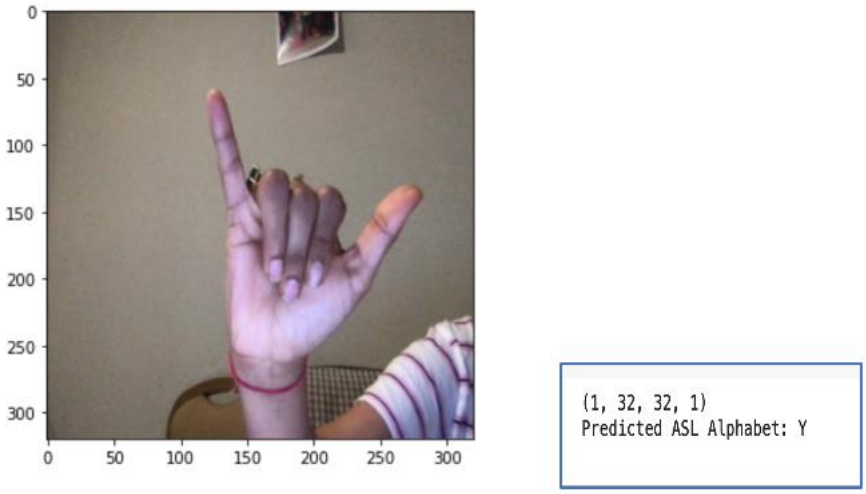


Figure 29: LeNet-5 ASL Alphabet Detection Result : Y class

d. ASL Word Recognition in Real-Time

For this project, we have done Keras implementation of LeNet-5 architecture for ASL Word Recognition as seen in Fig. 30 which contains 2 Convolution layer & 2 Max pooling layer, and then it was flattened. The next two dense layers are added which have an activation function like sigmoid and relu. Finally, a SoftMax classifier is used to give out final results.

```
[ ] import keras
    from keras.models import Sequential
    from keras.layers import Conv2D, Dense, MaxPool2D, Dropout, Flatten
    from keras.optimizers import Adam
    from keras.preprocessing.image import ImageDataGenerator
    from keras.callbacks import ReduceLROnPlateau
    from sklearn.model_selection import train_test_split
    import matplotlib.pyplot as plt
    import seaborn as sns
```

```
[ ] model = Sequential()
    model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu', input_shape=(64, 64,3)))
    model.add(MaxPool2D(strides=2))
    model.add(Conv2D(filters=64, kernel_size=(3,3), padding='valid', activation='relu'))
    model.add(MaxPool2D(strides=2))
    model.add(Flatten())
    model.add(Dense(256, activation='sigmoid'))
    model.add(Dense(84, activation='relu'))
    model.add(Dense(9, activation='softmax'))
```

```
[ ] model.summary()
```

```
[ ] import tensorflow as tf
```

```
opt = Adam(lr=0.00001)
model.compile(optimizer = opt , loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True) ,
              metrics = ['accuracy'])
```

Figure 30: LeNet-5 Keras Model for ASL

To better compare the performance of the model as part of our analysis, we have also implemented a Support Vector Machine for ASL Word Recognition. We configured the hyperparameters like C, kernel, and gamma as seen in Fig. 31.

```
param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
 ]
svc = svm.SVC()
clf = GridSearchCV(svc, param_grid)
clf.fit(X_train, y_train)
```

Figure 31: SVM Model for ASL

V. EXPERIMENTS

This section describes various experiments that we performed to analyze the performance of our proposed methodology for ASL gesture recognition on various datasets.

a. MNIST ASL ALPHABET DATASET

In this experiment, we used the MNIST ASL Alphabet dataset that contains the 24 ASL letters that are static in nature, and the remaining 2 letters are motion-based which are not part of the dataset. The class labels range from 0 to 25 corresponding to alphabets starting from 0 for the letter ‘A’ to 25 for the letter ‘Z’. While training the model we got an accuracy of 95.08% as shown in Fig. 32 in 20 epochs where steps per epoch were 10. The graph of the training accuracy was observed to be monotonically increasing with the number of epochs as seen in Fig. 33. When real-time predictions were tested, they performed well as can be seen in the sample demo in Fig. 29 where ‘Y’ was predicted accurately.

```
#train the data on lenet5 arch
model.fit(X_train ,Y_train, steps_per_epoch = 10, epochs = 20)
```

```
Epoch 1/20
10/10 [==============================] - 1s 26ms/step - loss: 3.1731 - accuracy: 0.0548
Epoch 2/20
10/10 [==============================] - 0s 21ms/step - loss: 2.9280 - accuracy: 0.1324
Epoch 3/20
10/10 [==============================] - 0s 18ms/step - loss: 2.6138 - accuracy: 0.2281
Epoch 4/20
10/10 [==============================] - 0s 18ms/step - loss: 2.2653 - accuracy: 0.3426
Epoch 5/20
10/10 [==============================] - 0s 17ms/step - loss: 1.8969 - accuracy: 0.4776
Epoch 6/20
10/10 [==============================] - 0s 18ms/step - loss: 1.5729 - accuracy: 0.5548
Epoch 7/20
10/10 [==============================] - 0s 19ms/step - loss: 1.3218 - accuracy: 0.6154
Epoch 8/20
10/10 [==============================] - 0s 20ms/step - loss: 1.1046 - accuracy: 0.6842
Epoch 9/20
10/10 [==============================] - 0s 20ms/step - loss: 0.9556 - accuracy: 0.7282
Epoch 10/20
10/10 [==============================] - 0s 18ms/step - loss: 0.8303 - accuracy: 0.7655
Epoch 11/20
10/10 [==============================] - 0s 18ms/step - loss: 0.7274 - accuracy: 0.7951
Epoch 12/20
10/10 [==============================] - 0s 18ms/step - loss: 0.6501 - accuracy: 0.8188
Epoch 13/20
10/10 [==============================] - 0s 18ms/step - loss: 0.5701 - accuracy: 0.8412
Epoch 14/20
10/10 [==============================] - 0s 19ms/step - loss: 0.5001 - accuracy: 0.8658
Epoch 15/20
10/10 [==============================] - 0s 18ms/step - loss: 0.4455 - accuracy: 0.8814
Epoch 16/20
10/10 [==============================] - 0s 18ms/step - loss: 0.3815 - accuracy: 0.9026
Epoch 17/20
10/10 [==============================] - 0s 18ms/step - loss: 0.3451 - accuracy: 0.9114
Epoch 18/20
10/10 [==============================] - 0s 18ms/step - loss: 0.2949 - accuracy: 0.9295
Epoch 19/20
10/10 [==============================] - 0s 18ms/step - loss: 0.2592 - accuracy: 0.9407
Epoch 20/20
10/10 [==============================] - 0s 18ms/step - loss: 0.2245 - accuracy: 0.9508
<tensorflow.python.keras.callbacks.History at 0x7fb5088c7950>
```

Figure 32: MNIST ASL Alphabets on LeNet-5 model Epoch Accuracies

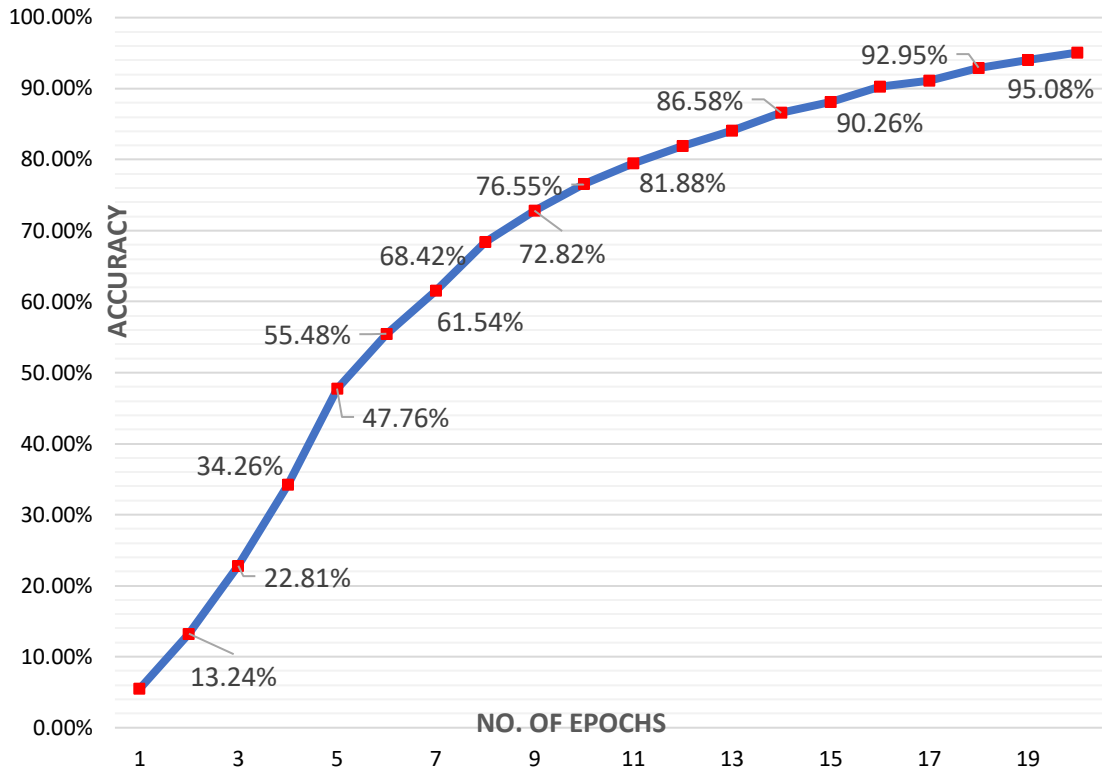


Figure 33: Training Accuracy for LeNet-5 ASL Alphabet Detection

b. NTU-RGB+D V/S LMC FOR SVM & LENET-5 MODEL

As part of this experiment, we tried to compare the performance of the NTU-RGB dataset and Leap Motion Controller dataset on LeNet-5 and SVM models.

1. NTU – RGBD+ Dataset

In this, the skeleton has 25 bone points and each of them had 3D coordinates. From this dataset, we considered the following classes for this experiment for ASL Action recognition: Hello (Salute), Hand Wave (Bye), Food (Eat).

a. LeNet-5 Results

When we trained the model on LeNet-5 architecture, for this particular dataset, we observed 86% training accuracy. The precision was best for class 2 i.e. eat gesture which was mostly focused on depth dimension i.e. Z coordinate.

```
[ ]  labels = np.argmax(yy, axis = 1)
     # print(labels.shape, labels)
     print(classification_report(y_test, labels, target_names =
                                 [' Hand Waving(Class 23)','Food Sign : Eat (Class 2)','Hello Sign (Salute) (Class 38)']))
```

	precision	recall	f1-score	support
Hand Waving(Class 23)	0.86	0.83	0.84	270
Food Sign : Eat (Class 2)	0.89	0.88	0.89	301
Hello Sign (Salute) (Class 38)	0.82	0.85	0.83	279
accuracy			0.86	850
macro avg	0.86	0.85	0.85	850
weighted avg	0.86	0.86	0.86	850

Figure 34: NTU RGB Classification Report for SVM

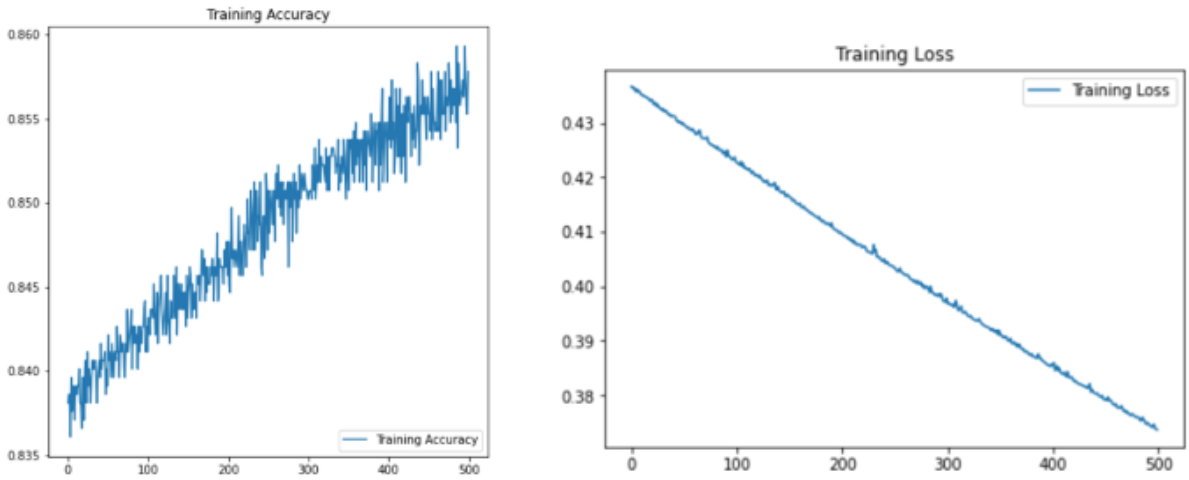


Figure 35: NTU RGB Training Accuracy and Loss Graph

As we can observe in the training accuracy and loss, the accuracy increased from 83.5% to 86% when learning_rate was 0.001 and optimizer was adam. The decrease in loss is also constant.

b. Support Vector Machine

In this experiment, we got an accuracy of 89% for the SVM model. This is better as compared to the LeNet-5 model's accuracy that was just 86%.

```
Classification report for -
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
                         {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                          'kernel': ['rbf']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0):
              precision    recall  f1-score   support

           0       0.89      0.88      0.88       270
           1       0.93      0.88      0.91       301
           2       0.86      0.91      0.88       279

    accuracy                           0.89       850
   macro avg       0.89      0.89      0.89       850
weighted avg       0.89      0.89      0.89       850
```

Figure 36: SVM Classification Report for NTU RGB Dataset

2. Leap Motion Controller Dataset

In this, the skeleton has 27 bone points and each of them had 3D coordinates. These points were focused on hand joints. From this dataset we considered the following classes for this experiment for ASL Action recognition: Please, Blue, Red, Yellow, Where, Stop, Water, Orange, Thanks performed majorly by the right hand.

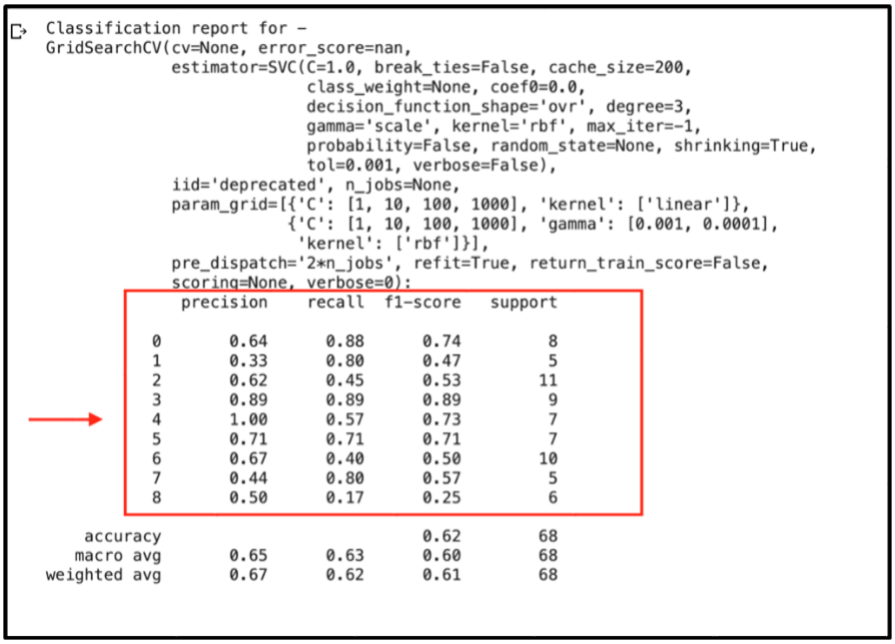
a. LeNet-5 Result

When we trained the model on this architecture, we got an accuracy of 15% which was very low. We tried to use different learning rates like 0.00001, 0.001, and 0.0001 but the accuracy still was extremely poor. We also tried to change the activation function used in the model architecture, but again it did not help in any way. We then tried to work with different

epoch steps and increased the number of epochs, it still did not perform. Because the skeleton point from LMC (Leap Motion Controller) was focused only on hand points the accuracy should have been much better than the NTU-RGB+D gesture dataset which focuses on full body. But from all these experiments, we deduced that the quality of the dataset from the LMC for LeNet-5 as an image classifier wasn't good and hence it did not do well on this methodology proposed in this report.

b. Support Vector Machine

When we trained the model on this architecture, we got much better results than the LeNet-5 model. The training accuracy for SVM is 65%. We observed from the classification report that most classes got precision around 60% and one of the classes was 100% classified.



```
Classification report for -
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
                         {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                          'kernel': ['rbf']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0):
              precision    recall  f1-score   support

           0       0.64      0.88      0.74         8
           1       0.33      0.80      0.47         5
           2       0.62      0.45      0.53        11
           3       0.89      0.89      0.89         9
           4       1.00      0.57      0.73         7
           5       0.71      0.71      0.71         7
           6       0.67      0.40      0.50        10
           7       0.44      0.80      0.57         5
           8       0.50      0.17      0.25         6

    accuracy                           0.62        68
   macro avg       0.65      0.63      0.60        68
weighted avg       0.67      0.62      0.61        68
```

Figure 37: Classification Report for SVM for 9 classes of LMC

c. OKAY SIGN FROM UNITY DATASET

As part of this experiment, we generated the Unity Game Engine based gesture dataset where a humanoid avatar closely resembling a human is performing an 'Okay' gesture in ASL. The dataset has 50 videos taken from varied angles in the 3D Unity Scene. This adds to the heterogeneity of the dataset. The key points of the skeleton as seen in Fig. 39 were collected from OpenPose and these were used to generate the temporal mappings. The skeleton file gives the X and Y coordinates and the confidence score of that bone point. These collective points were used to create a 2D matrix and then transformed into temporal mappings and passed to the image classifier. It was observed that the standalone OpenPose installation is too slow without GPU support for the given video. It took 58 minutes to process a 5 sec video and to generate the co-ordinates. The average length of each video of our dataset is around 10sec and there are 50 videos in total. So, the total time duration was 500 secs. It was taking too long to just get the bone points from the dataset. And furthermore, adding the time duration to train the model for temporal mapping and action recognition also needs to be accounted. So, making it overall a computationally expensive process. We tried using Google Colab for getting GPU support, but it does not support symbolic link in drive which was necessary for building the OpenPose project. Since, we aimed for real-time recognition, given the high processing time we kept this for future scope.

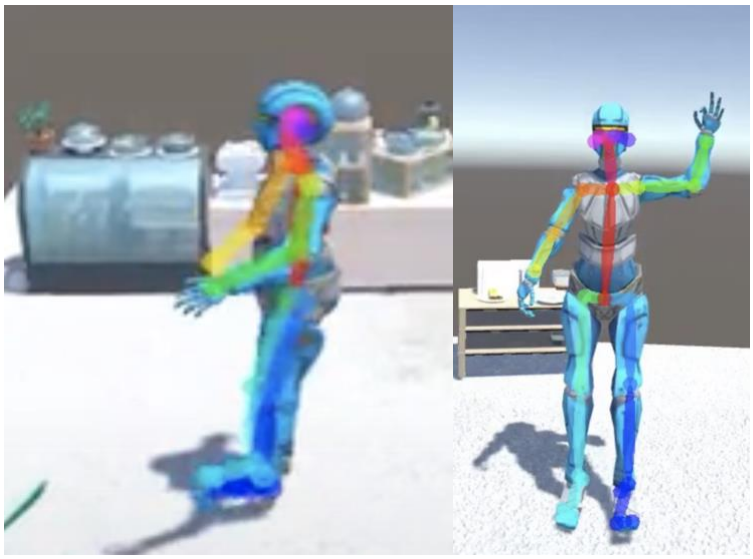


Figure 38: Output Skeleton visualized from OpenPose

video_000000000001_keypoints.json 〉 No Selection

```
1  {"version":1.3,"people":[{"person_id":[-1],"pose_keypoints_2d"
       [554.002,91.7185,0.863797,550.957,200.688,0.88472,450.822,
       .852359,651.037,212.445,0.731033,739.096,338.887,0.898874,
       .113,0.618613,318.418,680.419,0.699537,200.665,880.551,0.7
       .885,957.106,0.748847,536.218,76.9127,0.83607,577.412,77.0
       .688,1004.1,0.672653,580.327,998.315,0.744701,547.939,974.
```

Figure 39: Skeleton coordinates generated from OpenPose

d. IMPACT ON ACCURACY 2D VS 3D CO-ORDINATES

In this experiment, we tried to find the performance of the model when only 2D (X, Y) data points of the skeleton are provided instead of 3D (X, Y, Z). As can be seen in Fig. 38, Fig. 39, and Fig. 40 after removing the Z coordinate which maps to blue color in the RGB pallet, the temporal patterns remain the same, but the dominance of blue color is taken over by green.

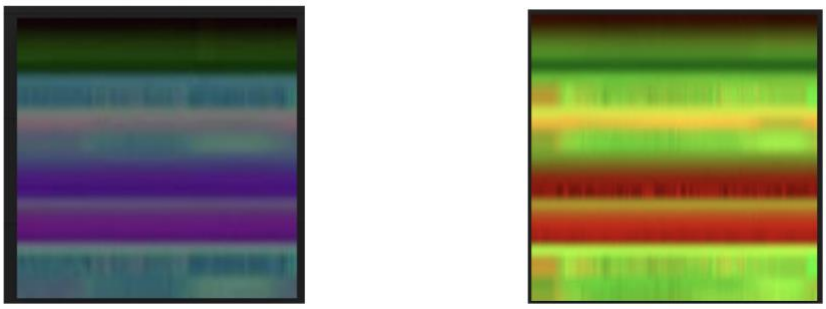


Figure 40: 3D v/s 2D temporal mapping for Class 2

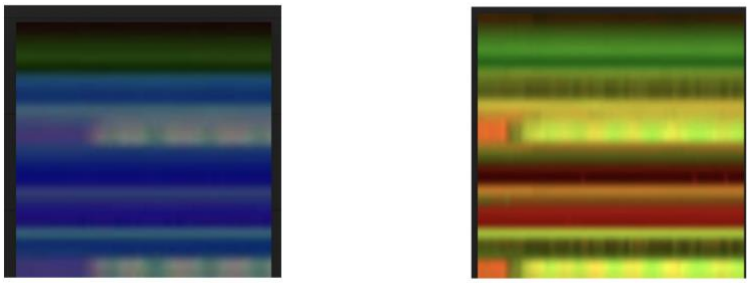


Figure 41: 3D v/s 2D temporal mapping for Class 23

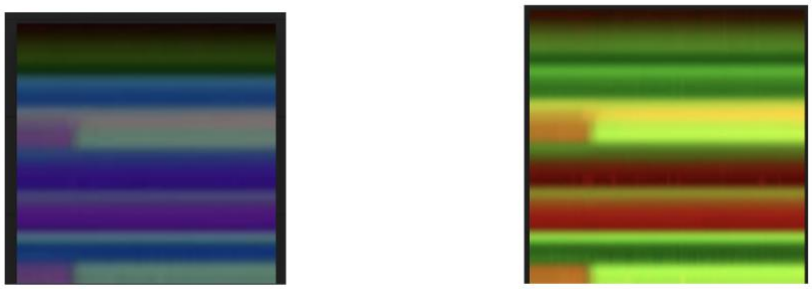


Figure 42: 3D v/s 2D temporal mapping for Class 38

When the model was trained with these new temporal images generated by 2D body joint coordinates, the accuracy wasn't impacted much. There was a slight increase of 1% in the accuracy leading it to 90% overall training accuracy as seen in Fig. 41. Also, the testing accuracy was found to be 89.52%. It was also observed that the precision of all the action

classes was almost the same. The reason that could be deduced from these results is that since the actions did not have much depth, so just considering 2D coordinates did not affect the performance of the model.

```
Classification report for -
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
                         {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                          'kernel': ['rbf']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0):
              precision    recall  f1-score   support

           0       0.89      0.94      0.91       270
           1       0.89      0.88      0.89       301
           2       0.91      0.86      0.88       279

    accuracy                           0.90       850
   macro avg       0.90      0.90      0.90       850
weighted avg       0.90      0.90      0.89       850
```

Figure 43: SVM Classification Report for 2D NTURGB Dataset

VI. CONCLUSION

The barrier in communication between people obstructs the normal way of living. This project creates a seamless experience for the deaf at the shopping mall by providing features that responds based on the recognized gesture from the customer. It helps customers to ask where to look for a certain item in-store, seek suggestions for clothing or ask for navigating to a washroom or request special assistance from a clerk.

We can use these models and add more gestures to the dataset to make the system more user-friendly and robust. We can also try to use more computer vision architecture for this methodology like LSTM that could be used to get the real-time ASL speech. Also, if facial features are combined with hand gestures then we could have a better understanding of the gestures. Whether one is a store owner or government, it's high time that we enforce a mechanism to maintain stores that are accessible for deaf customers.

REFERENCES

1. "Statistics on Voice, Speech, and Language," *National Institute of Deafness and Other Communication Disorders*, 01-Dec-2020. [Online]. Available: https://www.nidcd.nih.gov/health/statistics/statistics-voice-speech-and-language. [Accessed: 15-Dec-2020].
2. H. Lane, B. Bahan, and R. Hoffmeister, "3," in *A journey into the deaf world*, Dawn Sign Press, 1996.
3. "Americans with Disabilities Act," *U.S. Department of Labor*, 01-Dec-2020. [Online]. Available: https://www.dol.gov/general/topic/disability/ada. [Accessed: 06-May-2021].
4. P. Chang, "*Inclusive Communication Through Virtual And Augmented Reality Technology*," *ARPost*, 02-Oct-2018. [Online]. Available: https://arpost.co/2018/10/02/inclusive-communication-virtual-augmented-reality/. [Accessed: 16-Dec-2020]
5. "5 must-have apps for deaf and hard of hearing people in 2020," *Inclusive City Maker*, 22-Oct-2020. [Online]. Available: https://www.inclusivecitymaker.com/smartphone-apps-deaf-people-2020/. [Accessed: 16-Dec-2020].
6. "How Can Shopping Malls Be Accessible to People with Disabilities?," *Inclusive City Maker*, 20-Oct-2020. [Online]. Available: https://www.inclusivecitymaker.com/shopping-malls-accessible-people-with-disabilities/. [Accessed: 16-Dec-2020].
7. J. J. Bird, A. Ekárt, and D. R. Faria, "*British Sign Language Recognition via Late Fusion of Computer Vision and Leap Motion with Transfer Learning to American Sign Language*," Sensors, vol. 20, no. 18, p. 5151, Sep. 2020.
8. B. Ren, M. Liu, R. Ding, and H. Liu, "*A Survey on 3D skeleton-based Action Recognition Using Learning Method*," arXiv.org, 14-Feb-2020. [Online]. Available: https://arxiv.org/abs/2002.05907. [Accessed: 13-May-2021]
9. H. H. Pham, H. Salmane, L. Khoudour, A. Crouzil, P. Zegers, and S. A. Velastin, "*Spatio Temporal Image Representation of 3D Skeletal Movements for View-Invariant Action Recognition with Deep Convolutional Neural Networks*," *Sensors (Basel, Switzerland)*, 24-Apr-2019. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6514994/. [Accessed: 13-May-2021]

10. M. Rizwan, “LeNet-5 - A Classic CNN Architecture,” *engMRK*, 21-Apr-2020. [Online]. Available: https://engmrk.com/lenet-5-a-classic-cnn-architecture/. [Accessed: 15-Dec-2020].

11. “Support-vector machine,” Wikipedia, 07-May-2021. [Online]. Available: https://en.wikipedia.org/wiki/Support-vector_machine. [Accessed: 13-May-2021].

12. S. Ray, “*SVM: Support Vector Machine Algorithm in Machine Learning*,” Analytics Vidhya, 23-Dec-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/. [Accessed: 13-May-2021]

13. Unity Technologies, “Creating and Using Scripts,” Unity. [Online]. Available: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. [Accessed: 15-Dec-2020].

14. CMU-Perceptual-Computing-Lab, “CMU-Perceptual-Computing-Lab/openpose,” *GitHub*. [Online]. Available: https://github.com/CMU-Perceptual-Computing-Lab/openpose. [Accessed: 15-Dec-2020].

15. C. Li, Q. Zhong, D. Xie, and S. Pu, “Skeleton-based Action Recognition with Convolutional Neural Networks,” *arXiv.org*, 25-Apr-2017. [Online]. Available: https://arxiv.org/abs/1704.07595. [Accessed: 15-Dec-2020].

16. Jun Liu, Amir Shahroudy, Dong Xu, Alex C. Kot, Gang Wang, "Skeleton-Based Action Recognition Using Spatio-Temporal LSTM Network with Trust Gates", TPAMI, 2018.

17. Jun Liu, Gang Wang, Ling-Yu Duan, Kamila Abdiyeva, Alex C. Kot, "Skeleton-Based Human Action Recognition with Global Context-aware Attention LSTM Networks", TIP, 2018.

18. S. Laraba, M. Brahimi, J. Tilmanne, and T. Dutoit, “3D skeleton-based action recognition by representing motion capture sequences as 2D-RGB images,” Wiley Online Library, 21-May-2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1782. [Accessed: 5-March-2021]

19. Amir Shahroudy, Jun Liu, Tian-Tsong Ng, Gang Wang, "*NTU RGB+D: A Large Scale Dataset for 3D Human Activity Analysis*", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016 [PDF] [bibtex].

20. Jun Liu, Amir Shahroudy, Mauricio Perez, Gang Wang, Ling-Yu Duan, Alex C. Kot, "*NTU RGB+D 120: A Large-Scale Benchmark for 3D Human Activity Understanding*", IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2019.

21. Tecperson, “Sign Language MNIST,” Kaggle, 20-Oct-2017. [Online]. Available: https://www.kaggle.com/datamunge/sign-language-mnist. [Accessed: 13-May-2021]

22. R. Dias, “American Sign Language Hand Gesture Recognition,” Medium, 18-Dec-2019. [Online]. Available: https://towardsdatascience.com/american-sign-language-hand-gesture-recognition-f1c4468fb177. [Accessed: 13-May-2021]