American Sign Language Assistant

Presented To:

- Dr. Christopher Pollett
- Dr. Robert Chun
- Shruti Kothari

Presented By:

Charulata Lodha | Spring 2021

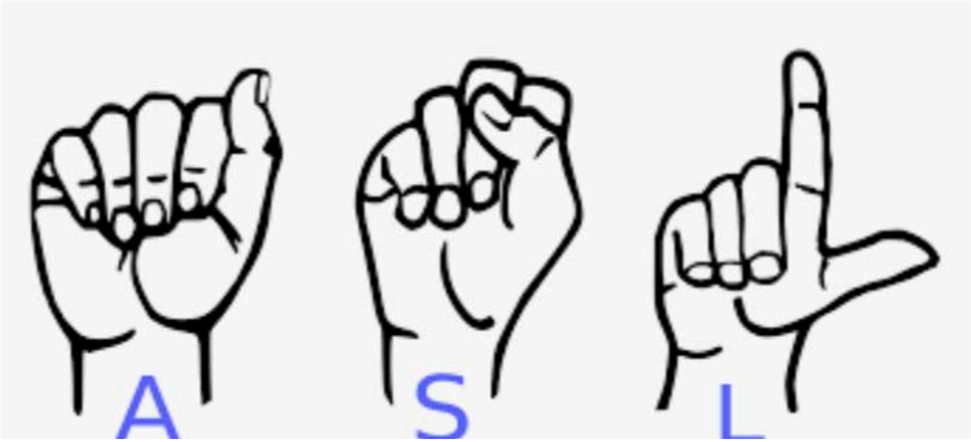
Department of Computer Science

San Jose State University

Agenda

Introduction

- Between 6 and 8 million people in the United States have some form of language impairment [1].
- Every situation can be stressing for a person with disabilities in a confined and busy place.
- The Americans with Disabilities Act (ADA) is a civil rights law that prohibits any kind of discrimination based on disability.
- The objective of this project is to build a digital assistance system to help deaf community in shopping center.



Introduction Continued…

The digital assistance system provides equivalent shopping experience to deaf community as the one with the ability to speak English

- Enable live communication between shop keeper and deaf customers
- Develop a base for building contactless and ASL signs based kiosk at shopping centers
- Leverage Convolution Neural Networks and Unity Game Engine

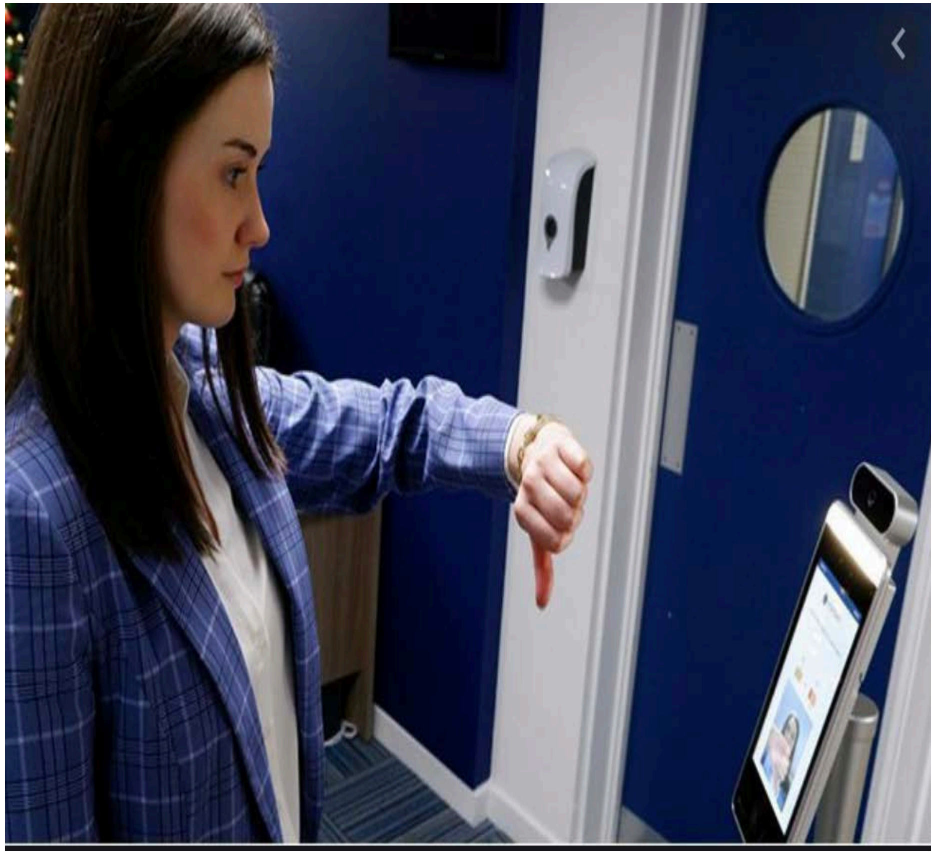


Image Courtey : https://www.kioskmarketplace.com/news/lamasatech-introduces-gesture-recognition-platform-for-sign-in-temp-screening-kiosks/

Background

- American Sign Language (ASL) is one of the leading sign language used in the U.S. [2].

- Existing accessibility technologies :
    - HoloHear : user speak ASL aloud to a 3D holographic model
    - TapSOS : connect with emergency services in a nonverbal way
    - Amplifier (FM systems) : to fully understand the advice of the sales assistant

Image Courtesy: https://akorbi.com/5-facts-you-didnt-know-about-american-sign-language/

Background : Shopping Center ASL

- ASL has hand gestures involves static as well as motion gestures.
- 3-Dimensional analysis is done with the perspective of building an AI model

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

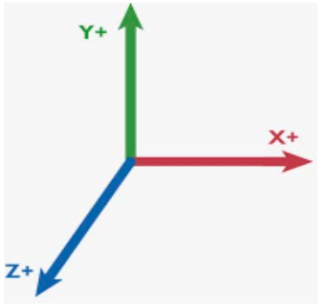


Figure 1: Red Sign in ASL

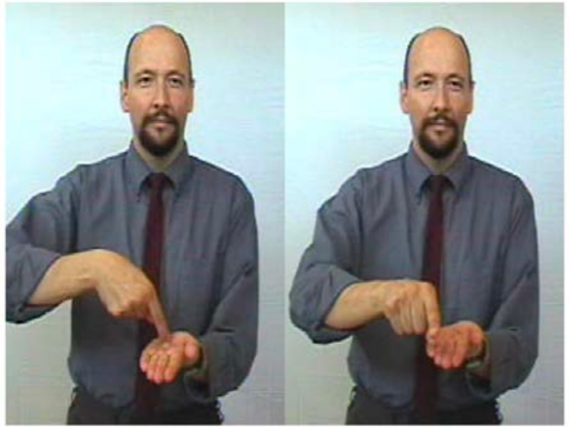


Figure 2: Pay Sign in ASL

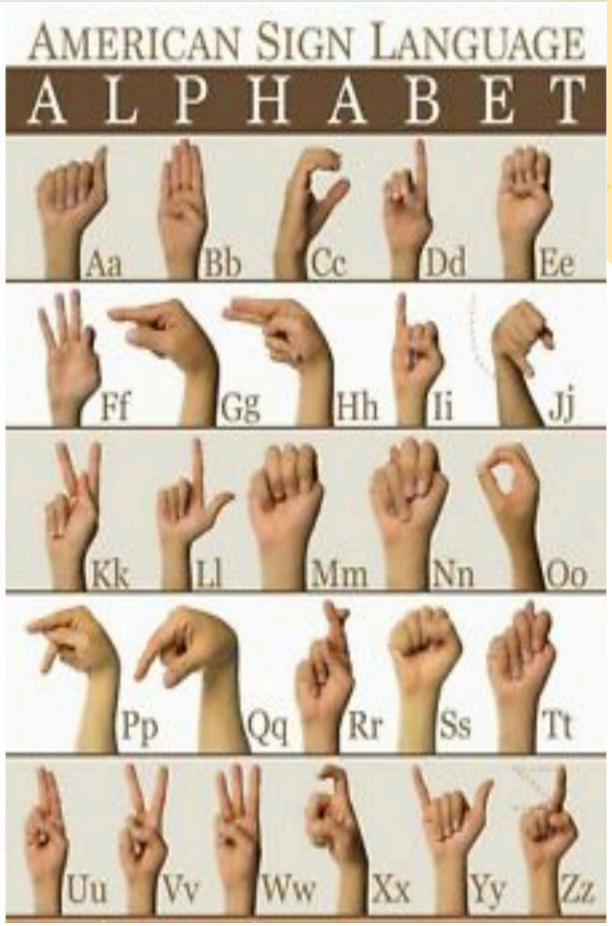
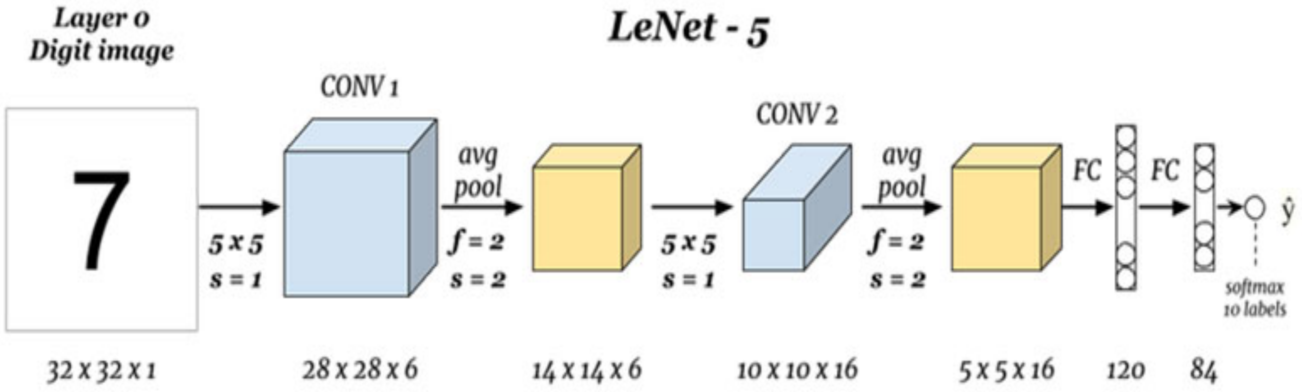


Figure 3. ASL Alphabets

Image Courtesy: https://akorbi.com/5-facts-you-didnt-know-about-american-sign-language/

Implementation : LeNet – 5 CNN Model

- Our project aims at finding visual patterns in temporal representation of ASL action sequences.
- Yann LeCun showed that minimizing the number of free parameters in neural networks can enhance the generalization ability of neural networks
- It is widely used for various image recognition problem.



Implementation :   LeNet – 5 CNN Architecture

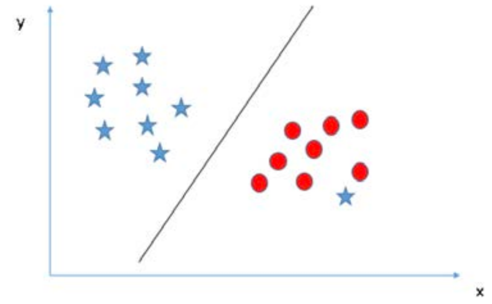
```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu', input_shape=(64, 64,3)))
model.add(MaxPool2D(strides=2))
model.add(Conv2D(filters=64, kernel_size=(3,3), padding='valid', activation='relu'))
model.add(MaxPool2D(strides=2))
model.add(Flatten())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

Implementation : LeNet – 5 CNN Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 14400)             0
_________________________________________________________________
dense (Dense)                (None, 256)               3686656
_________________________________________________________________
dense_1 (Dense)              (None, 84)                21588
_________________________________________________________________
dense_2 (Dense)              (None, 9)                 765
=================================================================
Total params: 3,728,401
Trainable params: 3,728,401
Non-trainable params: 0
_________________________________________________________________
```

Implementation : Support Vector Machine Machine Learning Model

- SVM is supervised machine learning algorithm
- It is widely used for classification and regression analysis.
- SVM can produce results on complex datasets that are smaller in size.
- Our datasets are temporal images that are very complex for general object recognition model.



SVM Hyperparameter	
C	penalty parameter to control error rate
Gamma	the kernel coefficient
Kernel	linear, sigmoid, rbf and poly

```
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 5: SVM Parameters for our model

Implementation : Unity Game Engine

We leveraged animations on humanoid avatar in Unity Game Engine for gesture dataset generation.

Typical Steps Involved :

1. Designing shop with items and humanoid avatars in 3D Unity Scene

2. Rescaling objects

3. Configuring Humanoid Rigs: Skeleton and Muscle Setting

4. Adding Animation Properties to Bone Points

5. Build custom gesture animation using Dopesheets and Curves

6. Designing Workflow/Animation State Diagram of events

8. IK Scriptwriting for object movement & camera placement

9. Light Settings in Unity

10. Mp4 video generation using Unity Recorder

Figure 6: ASL Gesture by Humanoid Avatar in Unity

Implementation : Unity Game Engine

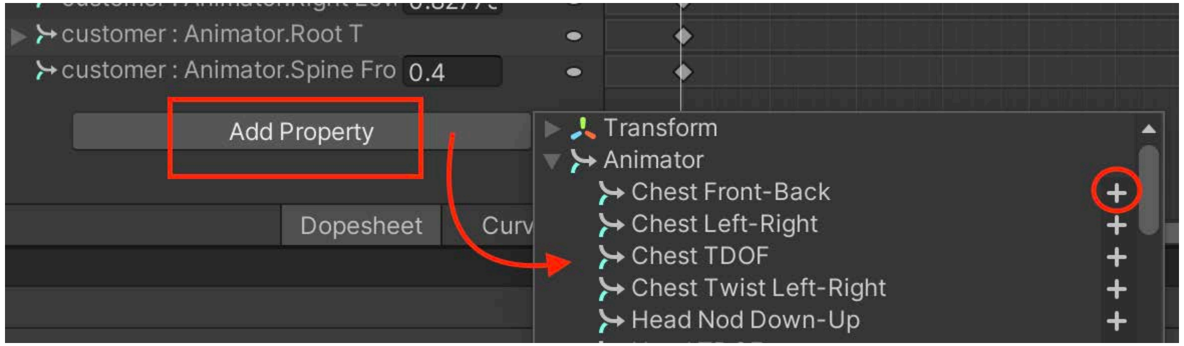


Figure 7: Bone Motion Property in Animation Configuration

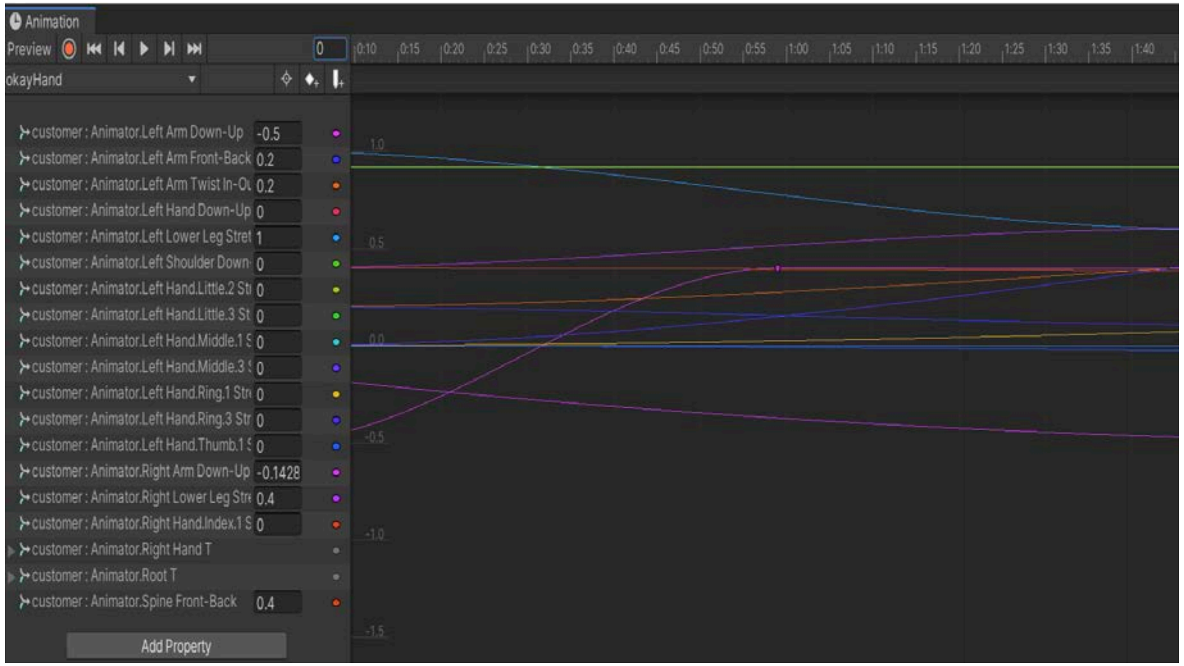


Figure 8: Okay Sign Curves Configuration

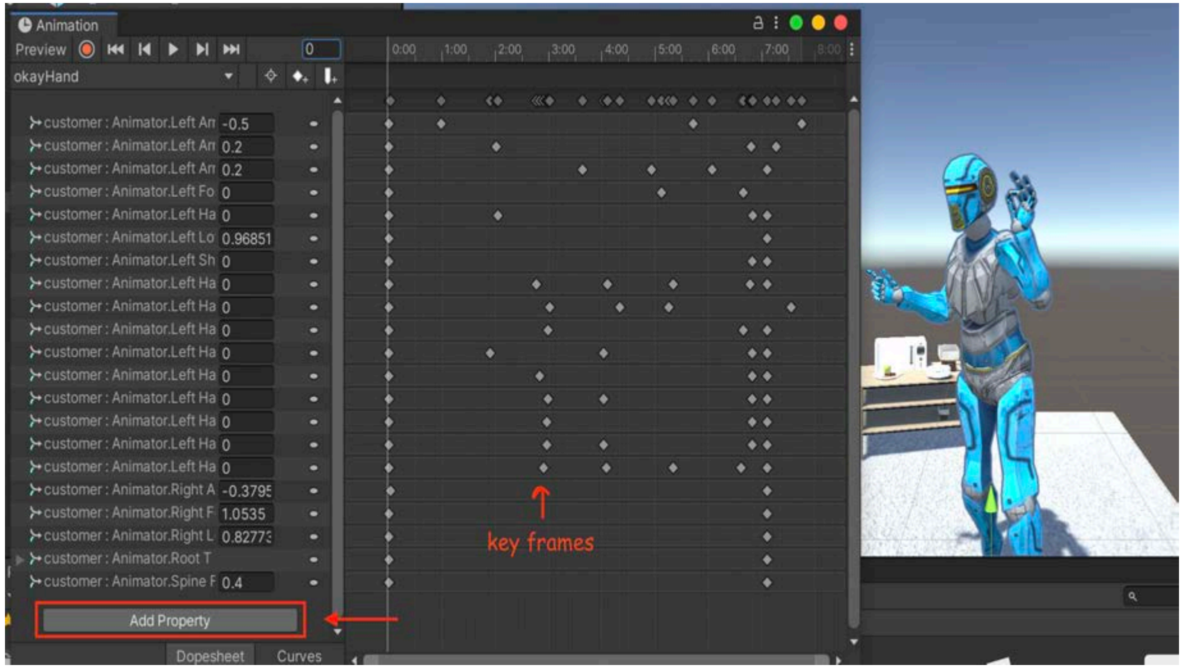


Figure 9: Okay Sign Dopesheet Configuration

Implementation : Unity Game Engine

Figure 10: Unity Scene : Customer doing Okay ASL Gesture Sign in a shop setting

Implementation : OpenPose

- OpenPose is a real time approach for multi-person key point detection : body, foot, hand, and facial key points.
- 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.
- It outputs 25 body part locations (x, y) and detection confidence (c) formatted as x0,y0,c0,x1,y1,c1....

test2out_000000000020_keypoints.json 〉 No Selection


```
1 {"version":1.3,"people":[{"person_id":[-1],"pose_keypoints_2d":
    [369.846,248.453,0.878195,352.889,390.735,0.577411,223.693,380.309,0.361781,138.859,476.929,0
    .146586,0,0,0,483.414,399.883,0.415472,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,338.581,218
    .429,0.892067,401.146,223.67,0.929188,299.364,230.204,0.737117,435.166,253.715,0
```

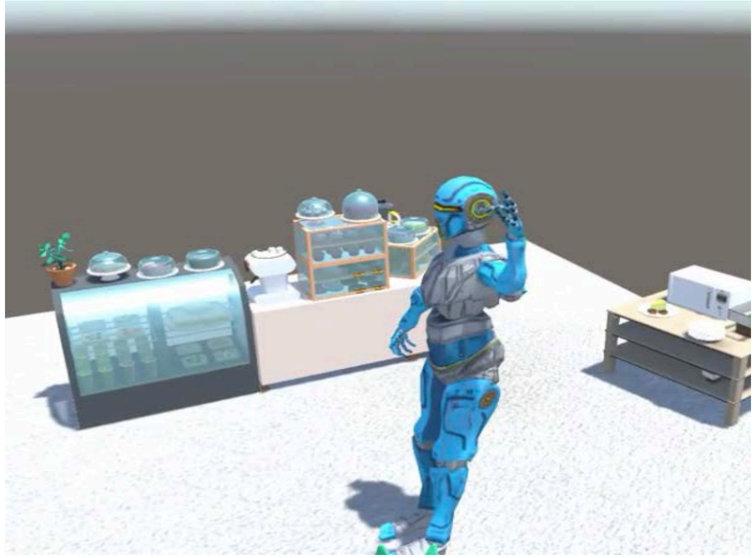



Figure 11: Unity Dataset Input Video to OpenPose

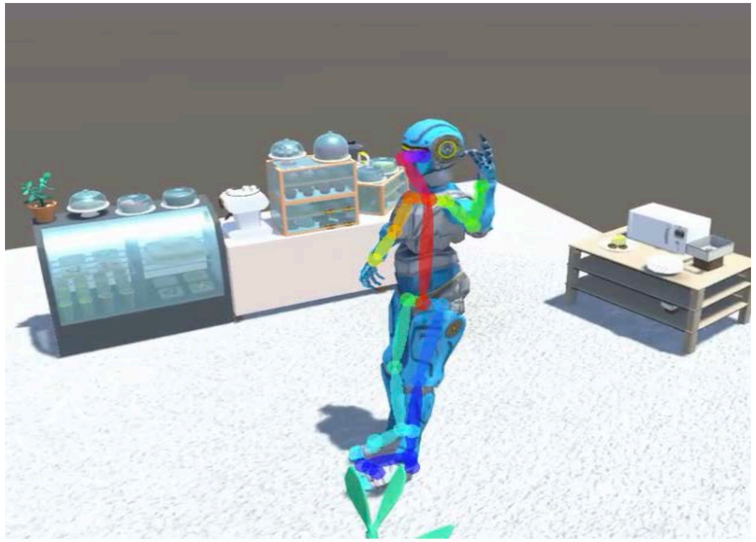
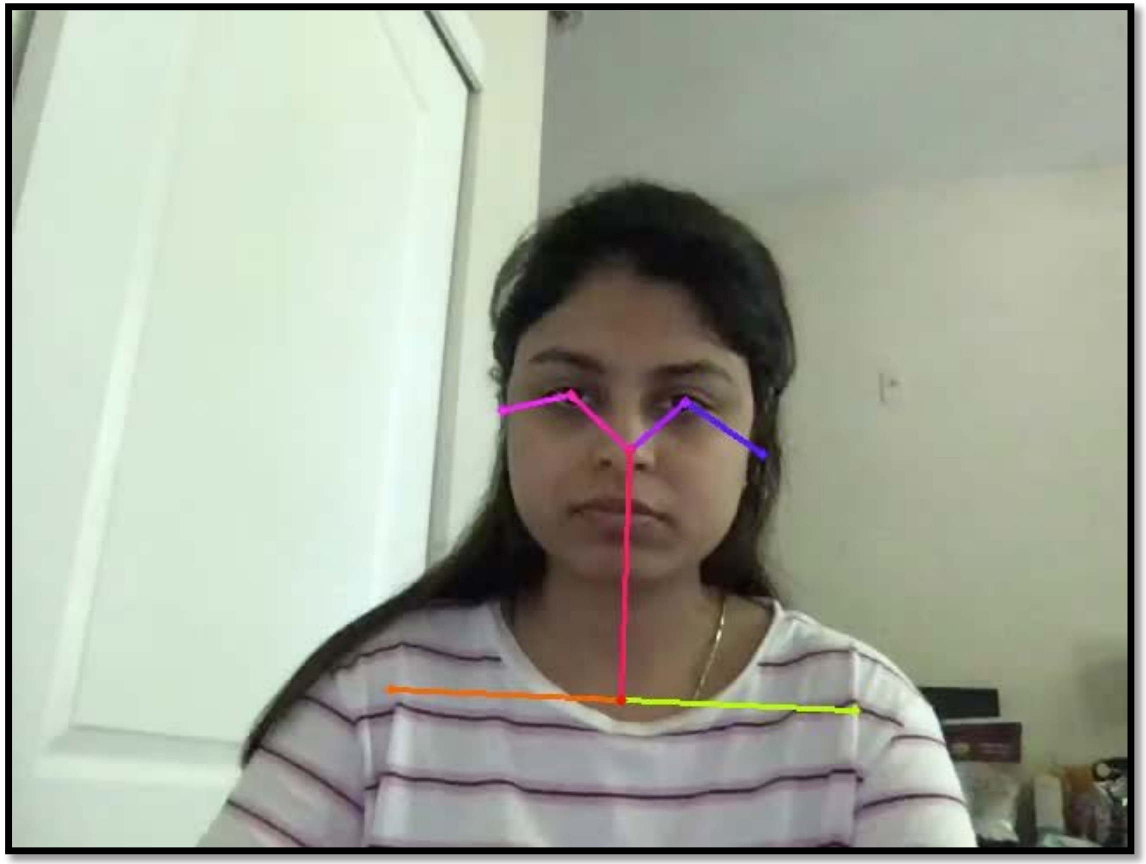
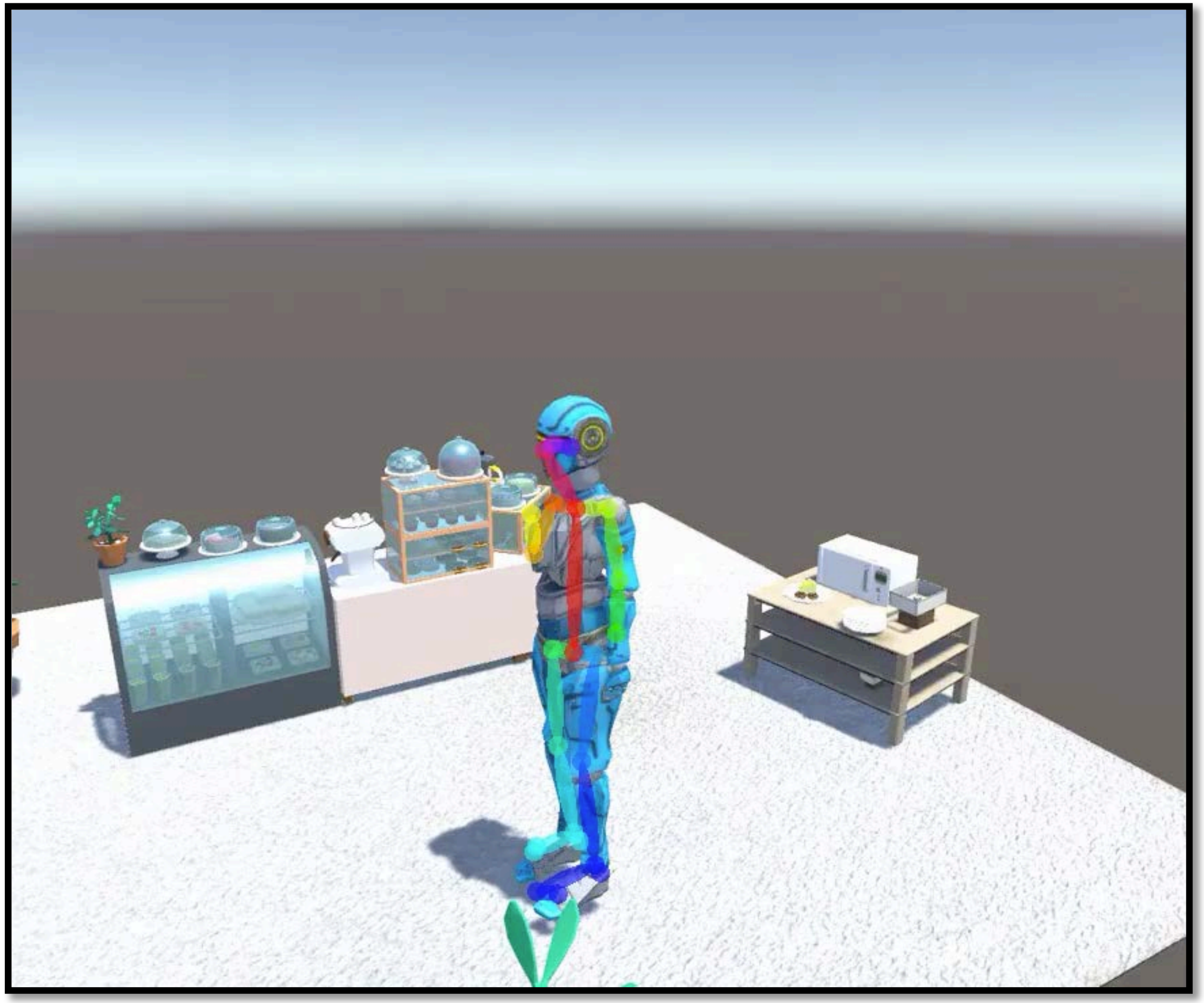


Figure 12: Output Video from OpenPose

Implementation : OpenPose





Implementation : Temporal Representation of 3D Skeletal Action Sequences

- The 3D skeleton is mapped into RGB color palettes.
- Its body invariant approach.
- We followed the approach proposed in [3] Sohaib, et al for transforming skeleton sequences into a temporal RGB image.
- This approach reduces the high dimensionality of action sequences to just image classification.

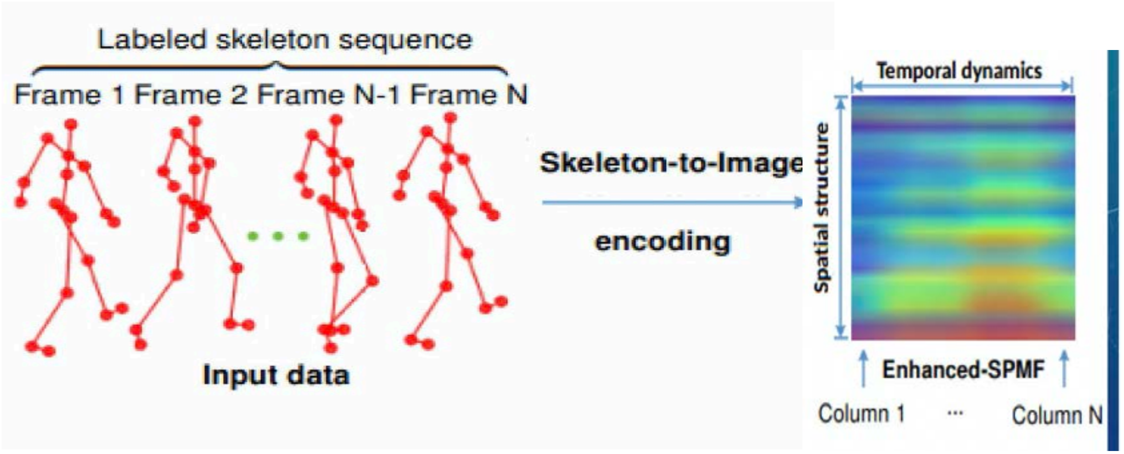


Figure 13: Skeleton to RGB Transformation

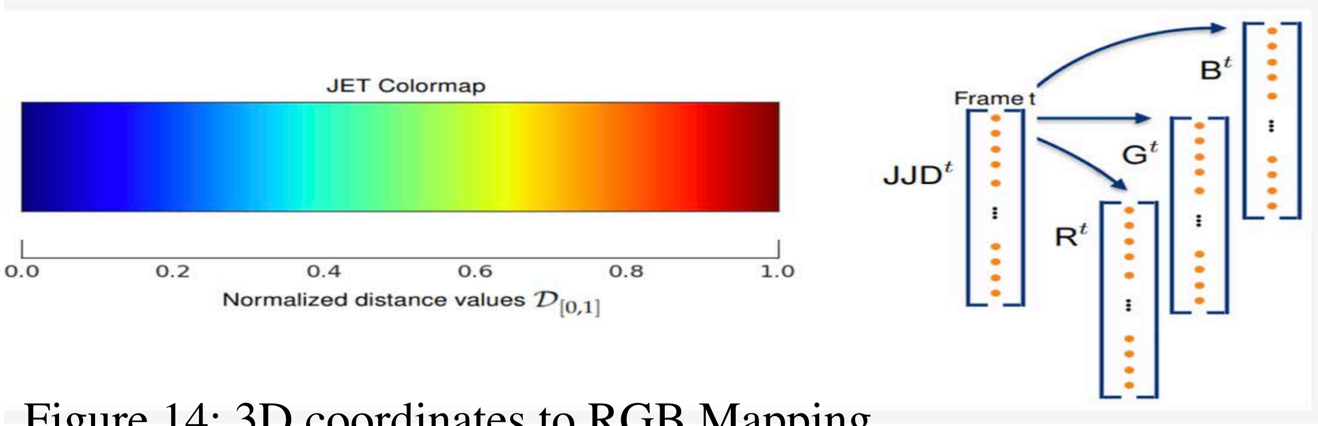


Figure 14: 3D coordinates to RGB Mapping

Implementation : Temporal Representation of 3D Skeletal Action Sequences

- **High Level Steps:**
  - Input : json/.skeleton skeleton files
  - Parse the input file to get the co-ordinates into numpy X,Y,Z matrix where each matrix has the shape = Number of Joint * Number of Frames
  - Normalized the X,Y,Z Matrix using Head Body Joint
  - RGB Channel Generation
  - Normalization of this to map to a range of 0 – 255 color value
  - Temporal Image Generation of size = 64*64

```python
for i in range(self.data.shape[0]):
    self.data[i,:, 0] -= self.data[self.norm, : , 0]
    self.data[i, :, 1] -= self.data[self.norm, :, 1]
    self.data[i, :, 2] -= self.data[self.norm, :, 2]
```

Figure 15: Normalization for 3D co-ordinate

```python
self.data[:, :, 0] = (self.data[:, :, 0] - min_x) / (max_x - min_x)
self.data[:, :, 1] = (self.data[:, :, 1] - min_y) / (max_y - min_y)
self.data[:, :, 2] = (self.data[:, :, 2] - min_z) / (max_z - min_z)

# for 2d
self.data[:, :, 2] = (self.data[:, :, 2]*0) / (max_z - min_z)
```

Figure 16: Normalization for 2D co-ordinate

Implementation : Temporal Representation Examples

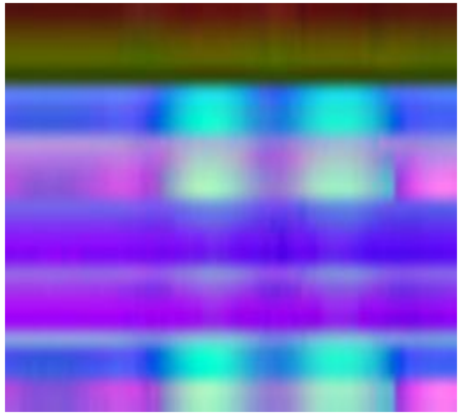


Figure 17: Human Jump (Sample output temporal image)

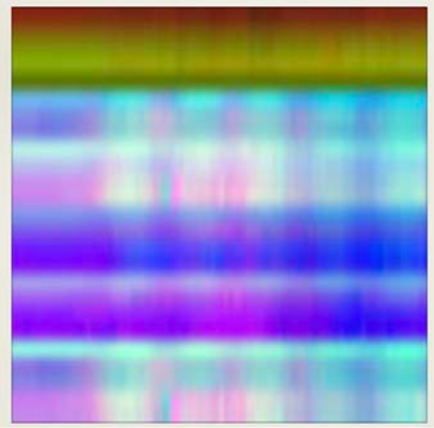


Figure 18: Human Hop (Sample output temporal image)

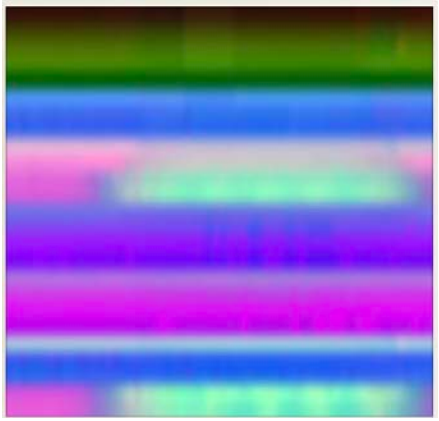


Figure 19: Human Hand Wave (Sample output temporal image)

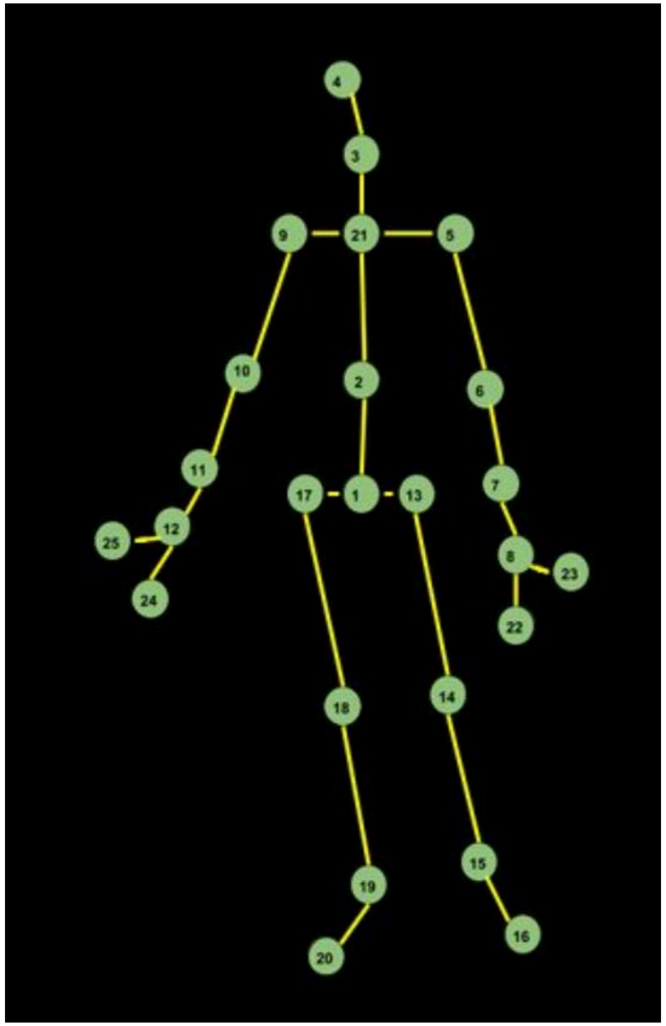
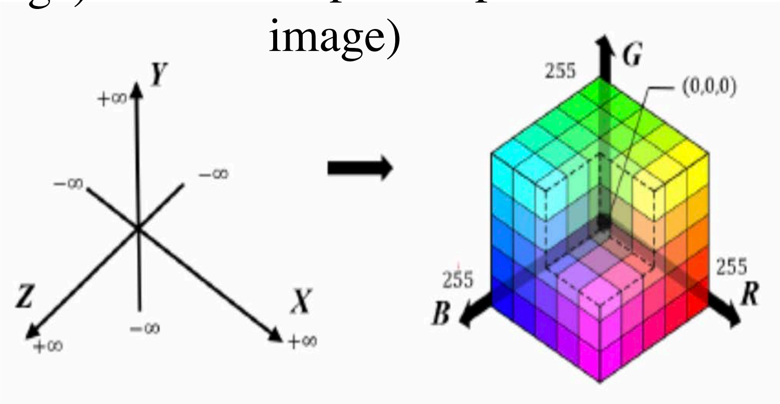


Figure 20: Skeleton Format

$$p_{\text{norm}} = \texttt{floor}\left(255 \times \frac{p - c_{\min}}{c_{\max} - c_{\min}}\right)$$



Related Work

- “British Sign Language Recognition via Late Fusion of Computer Vision and Leap Motion with Transfer Learning to American Sign Language” by Bird, J.J.; Ekárt, A.; Faria, D.R
  - fusion approach to multi-modality in sign language recognition
- “Skeleton-based Action Recognition with Convolutional Neural Networks” by Chao Li, Qiaoyong Zhong, Di Xie, Shiliang Pu
  - first to adapt Faster R-CNN to the task of skeleton-based temporal action detection
- “3D Skeleton-Based Action Recognition by Representing Motion Capture Sequences as 2D-RGB Images” by Sohaib Laraba, Med Brahimi, Joëlle Tilmanne and Thierry Dutoit
  - present a new representation of motion sequences Seq2Im for sequence to image) which projects motion sequences onto the RGB domain.

Image Courtesy: https://akorbi.com/5-facts-you-didnt-know-about-american-sign-language/

Design : Application Design

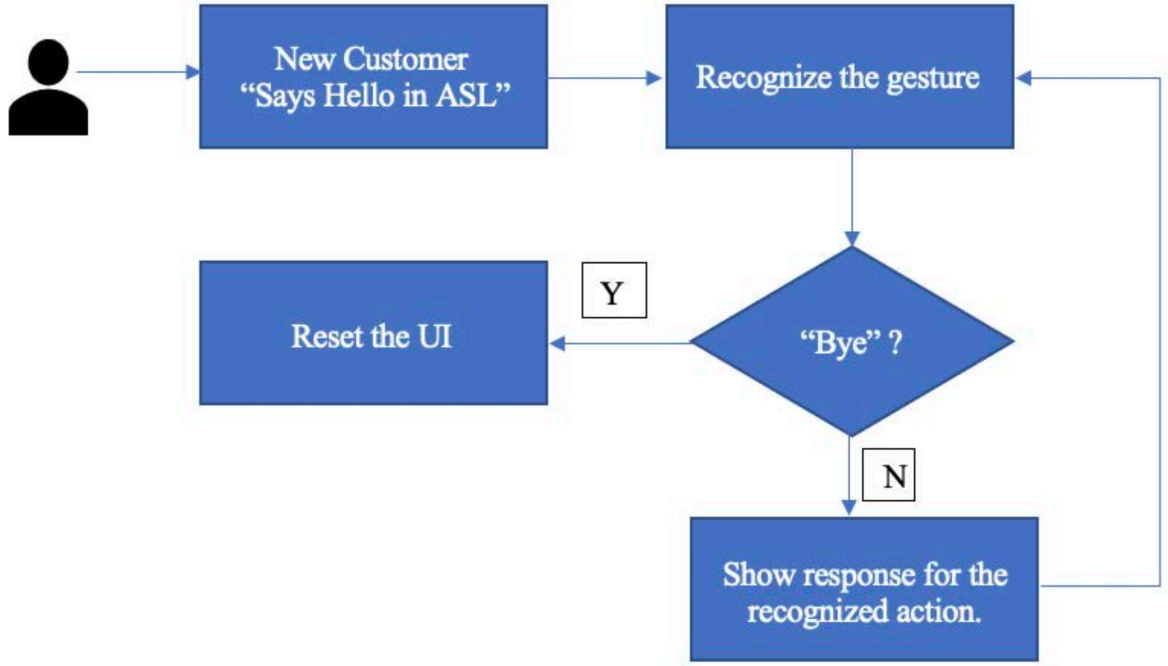


Figure 21 : Application Design

Design : Project Architecture

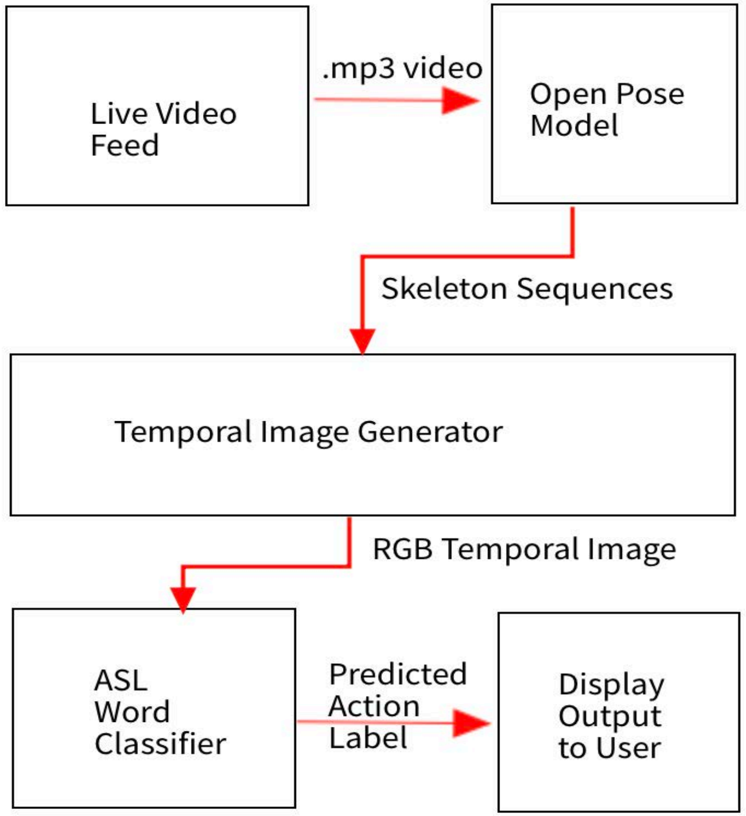


Figure 22 : Application Design

Datasets Overview

- MNIST ASL Alphabets Dataset
- Unity Okay Animation Dataset
- NTU RGB Action Recognition Dataset
- ASL Leap Motion Controller Dataset

Datasets : MNIST ASL Alphabets Recognition

- It matches closely with the classic MNIST.
- This dataset is provided as a .csv file that has 1 column for label and 784 columns for pixel values of the gesture image.

- Class label from 0 to 25 map to A-Z.
- Motion gestures J=9 or Z=25 excluded.
- Training data size = 27,455 and
  Test data size = 7,172

Dataset Sample Preview:

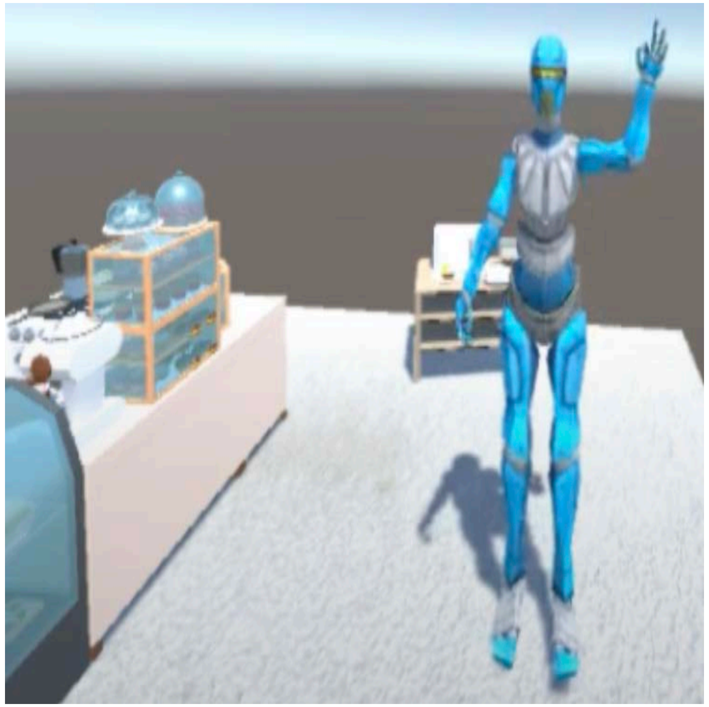
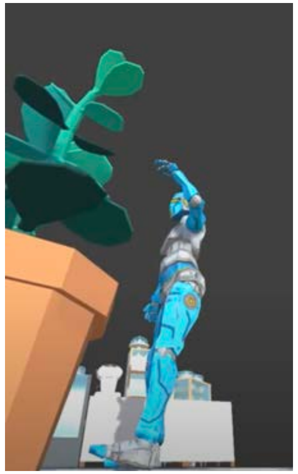
	label	pixel1	pixel2	pixel3	...	pixel781	pixel782	pixel783	pixel784
0	3	107	118	127	...	206	204	203	202
1	6	155	157	156	...	175	103	135	149
2	2	187	188	188	...	198	195	194	195
3	2	211	211	212	...	225	222	229	163
4	13	164	167	170	...	157	163	164	179
...	...	...	...	...	...	...	...	...	...
27450	13	189	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 23: MNIST ASL Dataset Preview

Datasets : Unity Okay Animation Dataset

- This is the synthetic video dataset generated by us using Unity Animation where a person is performing an ‘Okay’ sign in ASL .
- The dataset has 50 videos taken from varied angles in the 3D Unity Scene.
- Variety is added through different background contrast, various angles of camera, light settings and through addition of shop objects.

Datasets : NTU RGB+D Action Recognition Dataset

- It contains videos captured from Kinect V2 cameras providing RGB videos, 3D skeletal data, depth map sequences for each video sample.
- Class labels from 1 - 60 to represent respective action classes.
- Total dataset = 56,880 videos and .skeleton files

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 24: NTU RGB Action Classes

Datasets : NTU RGB Action Recognition Dataset

S018C001P041R001A102.skeleton

Class label

Skeleton Points(x,y,z)

Body joints

```
67
1
72057594037934351 0 1 1 0 0 0 0.1998573 -0.2021501 2
25
-0.1603807 0.006462337 3.412063 239.1637 207.8031 927.1428 565.0251 -0.2092761 0.04693484 0.9730155 -0.08509673 2
-0.142735 0.317171 3.353397 240.7749 173.8903 932.3832 467.1323 -0.2265078 0.04843735 0.9691491 -0.08425056 2
-0.1245192 0.6206408 3.281826 242.4364 139.1553 937.8688 367.192 -0.2436049 0.0616893 0.9604627 -0.1198433 2
-0.1732171 0.7506473 3.269492 236.8981 124.2077 922.1119 324.2775 0 0 0 0 2
-0.2891632 0.5207086 3.374737 224.9455 151.9475 886.8051 403.857 0.1884566 0.7486431 -0.6274622 -0.1015321 2
-0.3409766 0.2730556 3.486886 220.5545 179.8318 873.2759 484.1526 0.1131782 0.7911696 -0.1994244 -0.5669842 2
-0.3601164 0.04360022 3.508726 218.7969 203.9491 867.8653 553.7488 0.06113227 0.5744343 0.01482306 0.8161301 2
-0.3817433 -0.03365478 3.519233 216.6565 211.9947 861.572 576.9638 0.1482188 0.573288 -0.0248821 0.805452 2
0.002205719 0.4739555 3.217169 256.5992 154.5444 978.96 411.5305 -0.1241538 0.7933295 0.5251671 -0.2818041 2
0.05945465 0.196763 3.235404 263.0675 186.2579 997.3299 502.9835 0.01569892 0.9815378 0.1053612 0.1588586 2
0.05145715 -0.02871202 3.232117 262.1676 211.7426 994.5536 576.5731 0.0186449 -0.5835288 0.004429252 0.8118663 2
0.04317311 -0.08645608 3.240051 261.219 218.2497 991.7154 595.3612 0.01806303 -0.5794056 0.09620306 0.8091401 2
-0.2273161 0.009071837 3.407381 231.9532 207.5219 906.3223 564.1605 -0.04321315 -0.6873973 0.7194186 -0.08974711 2
-0.2179198 -0.3453481 3.546755 233.8631 244.1286 910.9375 669.8456 -0.1106592 -0.5280138 0.1502727 0.828477 2
-0.1807852 -0.6735729 3.706198 238.4653 275.1238 923.3652 759.0821 -0.1633541 -0.5204976 0.1608808 0.8225054 2
-0.2076803 -0.7639376 3.66709 235.5691 284.9297 915.1207 787.2019 0 0 0 0 2
-0.08982801 0.003812812 3.344495 246.5302 208.0788 948.7761 565.8752 -0.2272923 0.6508409 0.6593801 -0.2999372 2
-0.02175445 -0.3444034 3.428887 254.027 245.2426 969.7429 673.2144 0.04987678 0.8408145 0.141843 0.5200229 2
0.04655967 -0.6998987 3.614395 261.0714 279.4981 988.9349 771.7944 0.1392087 0.7820653 0.2089046 0.570398 2
0.03869141 -0.793726 3.584099 260.3096 289.7632 986.77 801.2334 0 0 0 0 2
-0.1291854 0.5458355 3.30203 242.0125 147.9243 936.4594 392.3806 -0.244032 0.05520767 0.9630563 -0.09961461 2
-0.3832865 -0.1154108 3.539151 216.7171 220.4287 861.5961 601.313 0 0 0 0 2
-0.3533302 -0.0449665 3.514429 219.5656 213.1766 869.9846 580.3987 0 0 0 0 2
0.02505606 -0.1676084 3.252883 259.1643 227.334 985.6303 621.58 0 0 0 0 2
0.007986417 -0.06139053 3.240583 257.249 215.4202 980.2557 587.1609 0 0 0 0 2
```

Figure 25: NTU RGB Skeleton for an Action Class

Datasets : ASL Leap Motion Controller Dataset

- This dataset contains 25 subjects performing 60 different signs of the ASL and includes more than 17,000 signs in total.

- 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 is idle for our project.

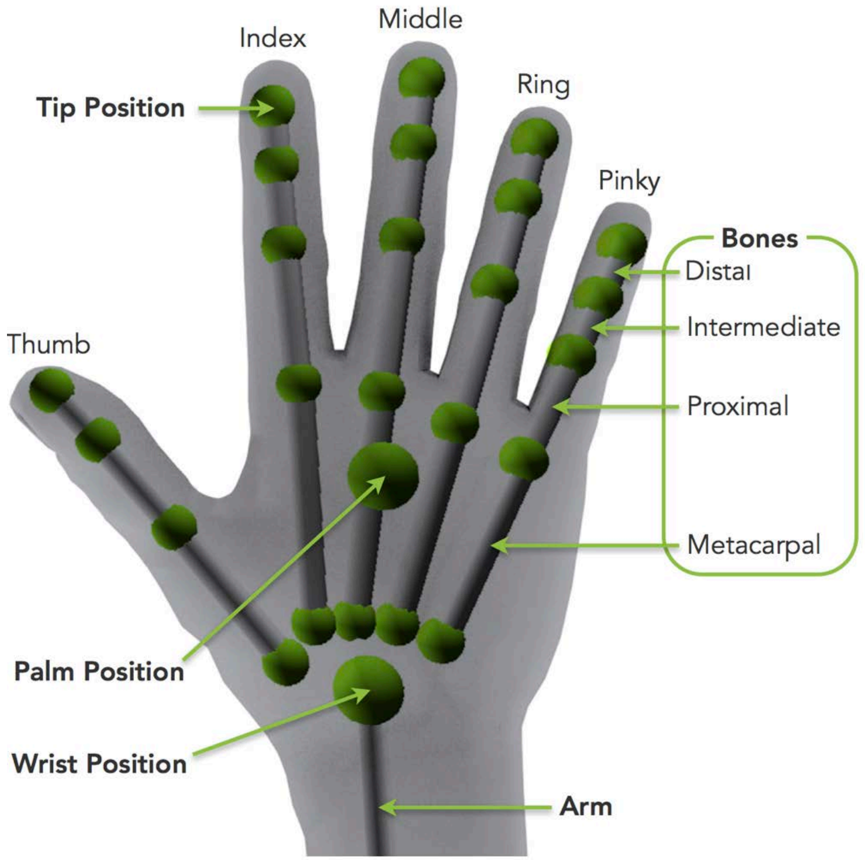


Figure 26: Bone Point Representation

Datasets : ASL Leap Motion Controller Dataset

- Every action has a separated dataset file for left and right hand.
- Each of these files represent the skeleton motion trajectory in 3D for that single hand.

	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 27: LMC Orange Left Hand Dataset Preview

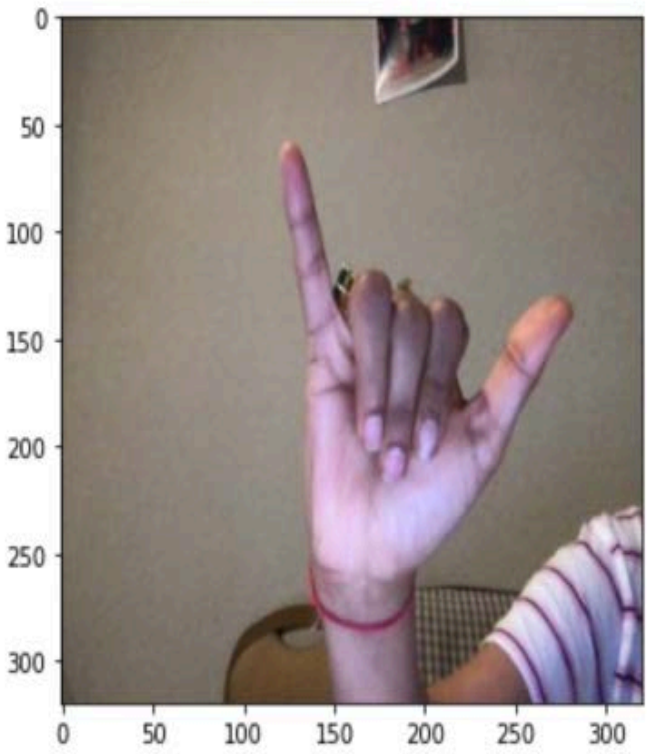
	Time	thumbProximal_L_X	thumbProximal_L_Y	thumbProximal_L_Z	thumbDistal_L_X
0	21:34:36.368 PM	-0.1097075	0.01622243	0.39756579999999997	-0.11329960000000001
1	21:34:36.395 PM	-0.10988230000000002	0.01601733	0.3980123	-0.1135501
2	21:34:36.425 PM	-0.1100047	0.01598163	0.39820559999999994	-0.11370119999999999
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

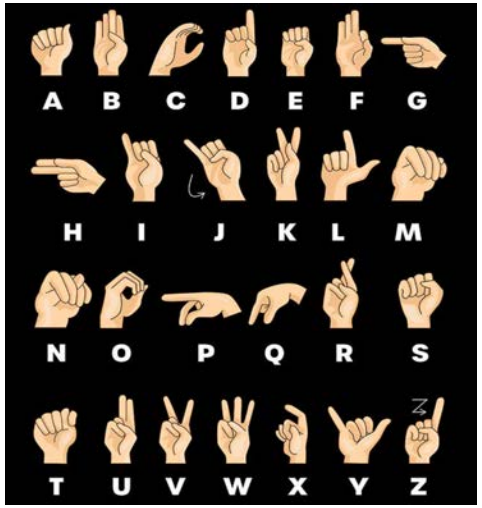
Figure 28: LMC Orange Right Hand Dataset Preview

Experiments and Results

- ASL Alphabet Recognition
- NTU-RGB+D on SVM & LeNet-5 model
- Leap Motion Controller on SVM & LeNet-5 model
- Impact on accuracy 2d vs 3d co-ordinates

Experiments : ASL Alphabets Recognition






```
(1, 32, 32, 1)
Predicted ASL Alphabet: Y
```

Figure 29: ASL Live Alphabet Recognition Preview

```
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
```

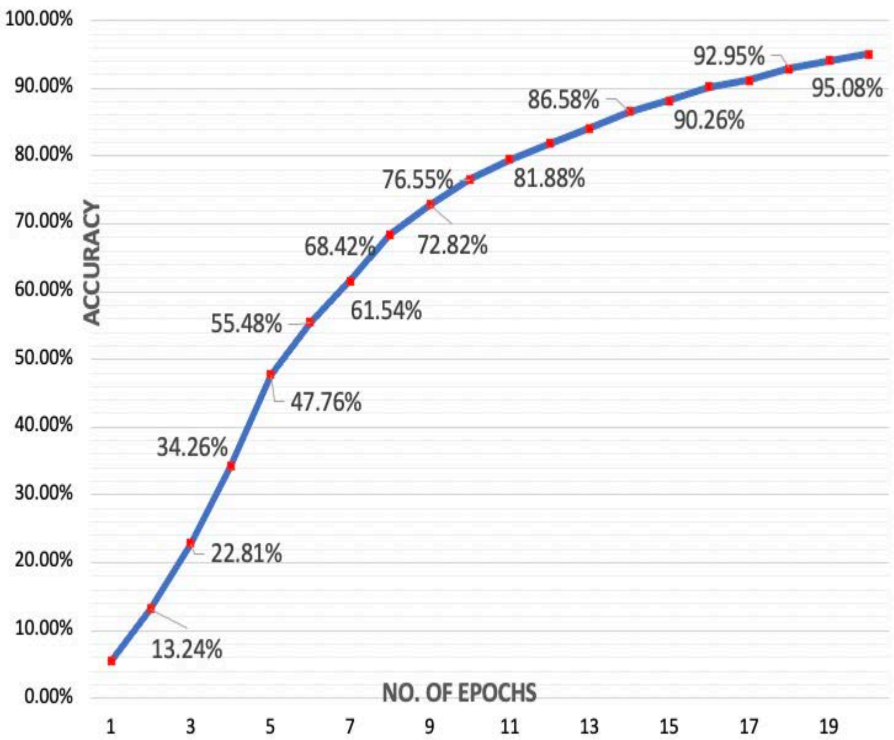
Figure 30: Le-Net 5 Model Summary

Results : ASL Alphabet Recognition

- The graph of the training accuracy was observed to be monotonically increasing with the number of epochs and accuracy was found to be 95.08% at the end of 20 epochs.
- Also, the loss was decreasing monotonically and reached to a final value of 0.2245

```
#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>
```



Experiments : NTU-RGB on LeNet-5 model

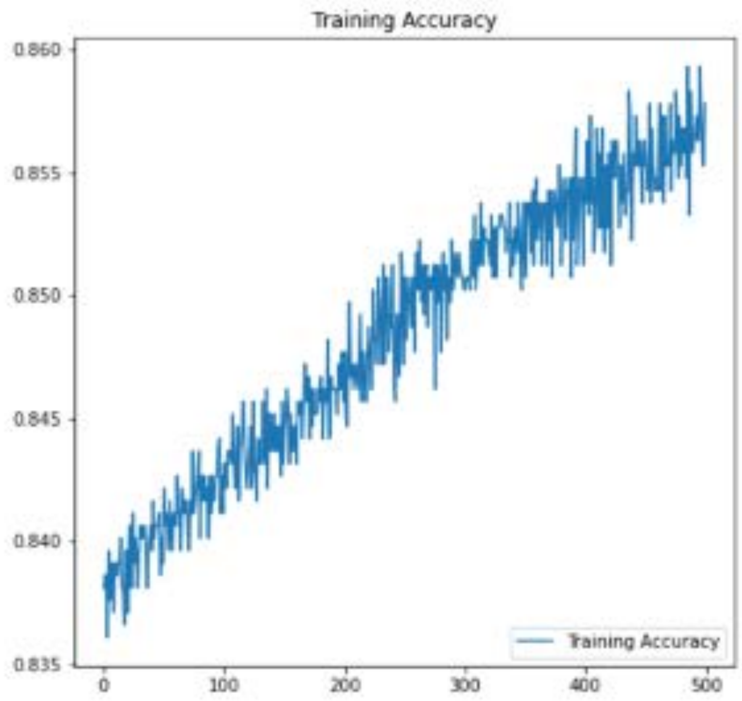
- When we trained the model on LeNet-5 architecture we observed 86% training accuracy for 3 classes whose gestures have complex motion in 3D space.
- The precision was best for eat gesture class which was most focused on depth dimension, i.e., Z coordinate.

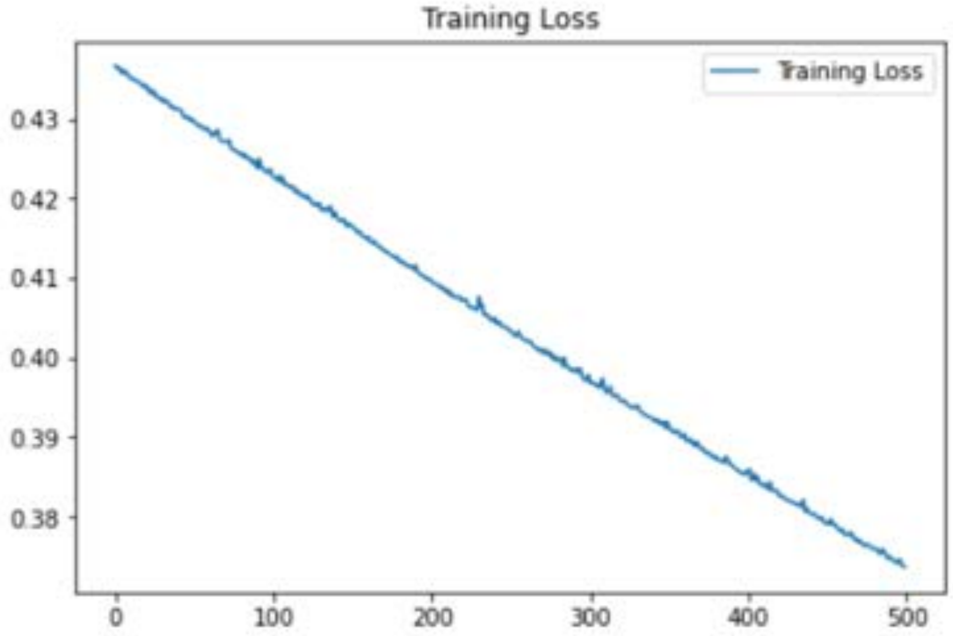
```
[ ] labels = np.argmax(yy, axis = 1)
    # print(labels.shape, labels)
    print(classification_report(y_test, labels, target_names =
                                ['Hand Waving(Class 23)','Food Sign : Eat (Class 2)','Hello Sign (Salute) (Class 38)']))
```

	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

Results : NTU-RGB+D on LeNet-5 Model

- As we can observe in the training accuracy and loss, the accuracy increased from 83.5% to 86% when learning rate was kept at 0.001 and Adam optimizer was used.
- The decrease in loss is also constant.





Experiments : NTU-RGB+D on SVM

- In this experiment we got accuracy of 89% for SVM model for the 3 classes.
- 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
```

Experiment : Leap Motion Controller Classes Used

- ASL Actions :

  Please, Blue, Red, Yellow, Where, Stop, Water, Orange and Thanks performed majorly by right hand.

Experiments : Leap Motion Controller on SVM

- When we trained the model on this architecture, we got much better results than the LeNet-5 model.
- The overall training accuracy for SVM is 65% for 9 classes.
- We observed from the classification report that most classes got precision around 60% and “where” of the class 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
```

Experiments: Leap Motion Controller on LeNet-5 model

- When we trained the model on this architecture, but it did not do well.
- We tried to use different learning rate like 0.00001, 0.001 and 0.0001 but the accuracy still was extremely poor around 13-15%.
- We also tried to change the activation function, epoch steps and number of epochs but accuracy was unaffected.
- As this dataset was totally focus on hand points, we deduced that the quality of dataset from the LMC for LeNet-5 as image classifier wasn’t good and hence it did not do well on this methodology proposed in this report.

Experiments : Impact on accuracy 2d vs 3d co-ordinates

- This experiment aimed at finding the performance of the model when only 2D (X, Y) data points of the skeleton are provided instead of 3D (X,Y, Z) using NTU RGB+D dataset.

- We observed that after removing the Z coordinate which maps to blue color in the RGB pallet, the temporal patterns remain same, but the dominance of colors is taken over by green and red with a huge contrast.

- Also, the accuracy wasn't impacted much. There was a slight increase of 1% in the accuracy leading it to 90% overall training accuracy.

```
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

Results : Impact on accuracy 2d vs 3d co-ordinates

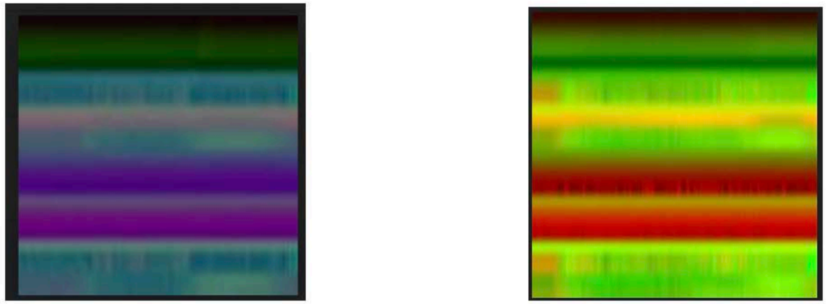


Figure 31: 3D v/s 2D temporal mapping for Class 2 : Food (Eat)

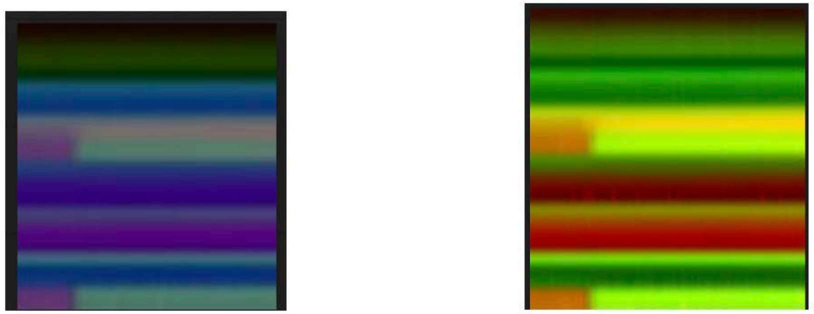


Figure 32: 3D v/s 2D temporal mapping for Class 38 : Hello (Salute)

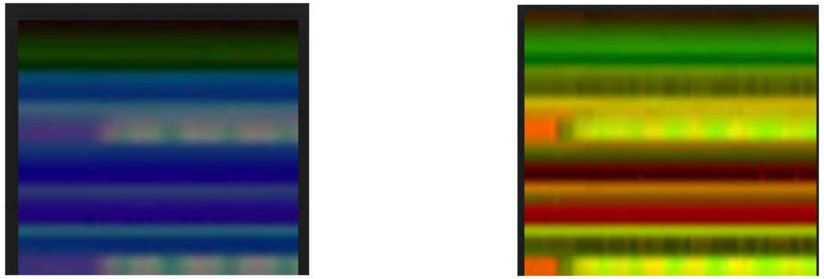


Figure 33: 3D v/s 2D temporal mapping for Class 23 - Bye

Output Temporal Images after transformation from Skeleton points to RGB image

Conclusions

- This project presented a prototype to helps customer to ask where to look for certain item in store, seek suggestions for clothing or ask for navigating to a washroom or request for special assistance from clerk.
- Accessibility technologies must provide real-time communication and be mandated in public places to aid with those hard of hearing or deaf just like we have strict enforcements for website.
- Also, with the spread of COVID-19 like virus, the demand for more and more contactless gesture-based technology is tremendously increasing.
- We can use these methodology and add more gesture to the dataset to make the system more user friendly and robust. We can also try to use other neural networks for this methodology like LSTM that could possibly be used to get the real time ASL speech.

References

- “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].
- H. Lane, B. Bahan, and R. Hoffmeister, “3,” in *A journey into the deaf world*, Dawn Sign Press, 1996.
- “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].
- 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]
- Support-vector machine,” Wikipedia, 07-May-2021. [Online]. Available: https://en.wikipedia.org/wiki/Support-vector_machine. [Accessed: 13-May-2021].

- Unity Technologies, “Creating and Using Scripts,” Unity. [Online]. Available: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. [Accessed: 15-Dec-2020].
- CMU-Perceptual-Computing-Lab, “CMU-Perceptual-Computing-Lab/openpose,” GitHub. [Online]. Available: https://github.com/CMU-Perceptual-Computing-Lab/openpose. [Accessed: 15-Dec-2020].
- 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]