

American Sign Language Assistant

A Project Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the

Class CS 297

By

Charulata Lodha

December 2020

ABSTRACT

We are developing a prototype computer vision system to help the deaf and mute communicate in a shopping setting. Our goal system would use video feeds to recognize American Sign Language (ASL) gestures and notify shop clerks of deaf and mute patrons' intents. Our prototype will operate on videos created in Unity of 3D humanoid models in a shop setting performing ASL signs.

This semester we wrote four short programs to help us towards the development of our final system. The first program was written for recognition of American Sign Language alphabets using Lenet-5 convolution neural network. Another program was written to make a humanoid avatar do sign language gestures in a shop setting using Animations in Unity. After this, a synthetic dataset creation C# program was used to capture the humanoid avatar doing gestures from different angles in Unity. Lastly, OpenPose, an open-source real-time pose estimation system, was used to detect the key points of the human body for the human gesture video dataset. Along with this a research paper was studied to understand how to efficiently recognize and classify the sign gestures performed by the human in a video using the bone key points information generated from OpenPose.

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

I. INTRODUCTION

In this technologically advanced world, we must utilize the power of artificial intelligence to solve some challenging real-life problems. One of the major issues that the world is still trying to cope up with is establishing an efficient way for communication between people. Between 6 and 8 million people in the United States have some form of language impairment [1]. American Sign Language (ASL) is the leading minority language in the U.S. after the "big four": Spanish, Italian, German, and French [2]. ASL is not sign language English. ASL is a visual language and the signs are used to convey ideas and concepts rather than actual words. As a result, there comes an undeniable communication barrier between the ASL and English speaking population. So, this project aims at building an accessibility technology using convolution neural networks that will recognize the American Sign Language gestures in a video stream by deaf and mute people and convert it into English text.

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 like the one with the ability to speak English.

The existing accessibility technolog[CL1] y shows an innovative application of Augmented Reality, Machine Learning, Artificial Intelligence, and Human-Computer Interaction to better accommodate mute and deaf people. In 2016, the HoloHear team prototype an augmented reality app using specifically the HoloLens. When a user would speak aloud, a 3D holographic model appeared to translate the sentence in American Sign Language in real-time [3]. Tap SOS app allows a person who is mute and deaf to connect with emergency services in a

nonverbal way and won the 2018 Digital Health Award [4]. Today shops provide amplifiers to deaf people to fully understand the advice of the sales assistant [5].

This project aims at 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 convolution neural networks.

The rest of the paper is organized as follows: In Section II, we talk about our first Deliverable 1 which depicts the ASL alphabets recognition using LeNet-5 convolution neural network. In Section III, we talk about Deliverable 2 and demonstrate how to create gestures on a humanoid avatar using Animations in Unity. Then we describe our application of gesture animation on humanoids in a shop setting in Unity and record a .mp4 movie of the scene. Then in Section IV, we describe how can we generate a synthetic 50 video dataset for any given human gesture using Animation in Unity as part of Deliverable 3. This dataset is created by placing the camera randomly in 3D space to record the subject i.e. our humanoid model from different angles to get the varied dataset of a gesture. In Section VI, we demonstrated how we used OpenPose to generate a new dataset that has the bone key points marking on the output video and 3d key points in an output JSON file. Finally, we conclude the paper in Section VI by talking about progress for the writing project this semester and how this work prepares us and acts as a base for the writing project in the next semester.

II. DELIVERABLE 1 : AMERICAN SIGN LANGUAGE ALPHABETS RECOGNITION USING LENET-5

The first deliverable was to develop a neural network to recognize images of the American Sign Language (ASL) alphabet. This deliverable will help in building a part of our final system of the writing project next semester that is capable of recognizing ASL alphabets gestures in a video feed. Our other deliverables this semester will focus on ASL words recognition. The ASL alphabets have 2 motion hand gestures for J and Z as Figure 1. The rest of the 24 alphabet hand gestures are static. In this deliverable, only static hand gesture recognition is done using convolution neural networks.

The LeNet-5 architecture has total of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, then two fully connected layers, and lastly a softmax classifier [6] as shown in Figure 2. This architecture was used for the implementation of our model



Figure 1. ASL Alphabet

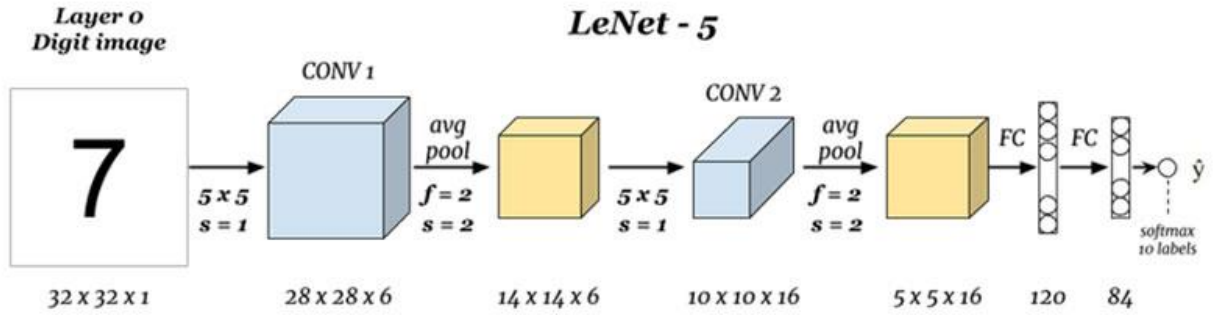


Figure 2. LeNet-5 Architecture

The model is trained and tested on the Sign Language MNIST dataset. The data is in CSV format with labels and pixel values in single rows. The training dataset has 27455 images and testing has 7172 images. The first column represents the label value for each alphabetic letter A-Z i.e. a value from 0-25. The 28*28 image is converted to 784-pixel cells in the row having grayscale values between 0-255 as shown in Figure 3.

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 3. Train Dataset Preview

An open-source Keras library is used to build the LeNet-5 architecture as shown in Figure 4. The model takes 32*32 input size image and our dataset 784 pixels i.e. 28*28. So, padding the dataset images by 2 pixels will give us an input image of size 32x32.

```

▶ model = Sequential()
#Layer 1
#Conv Layer 1
model.add(Conv2D(filters = 6, kernel_size = 5, strides = 1, activation = 'relu', input_shape = (32,32,1)))
#Pooling layer 1
model.add(MaxPooling2D(pool_size = 2, strides = 2))

#Layer 2
#Conv Layer 2
model.add(Conv2D(filters = 16, kernel_size = 5, strides = 1, activation = 'relu', input_shape = (14,14,6)))
#Pooling Layer 2
model.add(MaxPooling2D(pool_size = 2, strides = 2))

#Flatten
model.add(Flatten())

#Layer 3
#Fully connected layer 1
model.add(Dense(units = 120, activation = 'relu'))

#Layer 4
#Fully connected layer 2
model.add(Dense(units = 25, activation = 'relu'))

#Layer 5
#Output Layer
model.add(Dense(units = 25, activation = 'softmax'))
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])

```

Figure 4. LeNet-5 Keras Model

There are 54,367 trainable parameters as seen in the model summary in Figure 5. In both the fully connected layer relu is used as an activation function. Finally, the output has a value from 0-25 corresponding to alphabet number starting from 0 for letter ‘A’ to 25 for letter ‘Z’.

Layer (type)	Output Shape	Param #
conv2d_26 (Conv2D)	(None, 28, 28, 6)	156
max_pooling2d_26 (MaxPooling)	(None, 14, 14, 6)	0
conv2d_27 (Conv2D)	(None, 10, 10, 16)	2416
max_pooling2d_27 (MaxPooling)	(None, 5, 5, 16)	0
flatten_13 (Flatten)	(None, 400)	0
dense_39 (Dense)	(None, 120)	48120
dense_40 (Dense)	(None, 25)	3025
dense_41 (Dense)	(None, 25)	650
=====		
Total params: 54,367		
Trainable params: 54,367		
Non-trainable params: 0		

Figure 5. LeNet 5 Model Summary

When 10 epochs were run, the model gave an accuracy of 97.91% as shown in Figure 6. The output was saved as a csv file having the predictions stored corresponding to the test dataset.

```

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

□ Epoch 1/10
  10/10 [=====] - 0s 29ms/step - loss: 0.5674 - accuracy: 0.8466
  Epoch 2/10
  10/10 [=====] - 0s 26ms/step - loss: 0.4756 - accuracy: 0.8706
  Epoch 3/10
  10/10 [=====] - 0s 25ms/step - loss: 0.3989 - accuracy: 0.8917
  Epoch 4/10
  10/10 [=====] - 0s 24ms/step - loss: 0.3342 - accuracy: 0.9139
  Epoch 5/10
  10/10 [=====] - 0s 27ms/step - loss: 0.2846 - accuracy: 0.9296
  Epoch 6/10
  10/10 [=====] - 0s 25ms/step - loss: 0.2373 - accuracy: 0.9444
  Epoch 7/10
  10/10 [=====] - 0s 25ms/step - loss: 0.1997 - accuracy: 0.9543
  Epoch 8/10
  10/10 [=====] - 0s 25ms/step - loss: 0.1682 - accuracy: 0.9639
  Epoch 9/10
  10/10 [=====] - 0s 26ms/step - loss: 0.1431 - accuracy: 0.9728
  Epoch 10/10
  10/10 [=====] - 0s 24ms/step - loss: 0.1209 - accuracy: 0.9791
  <tensorflow.python.keras.callbacks.History at 0x7f2195f0cfd0>

```

Figure 6. LeNet-5 Model Training

To perform a prediction, an input images was given as pixel values of size 28*28 as shown in Figure 7 where the sample image is of letter ‘Y’.

```

▶ input="162 165 168 171 173 174 177 179 180 180 181 181 182 183 183 18
▶ arr = input.split("\t")
  inputimgarray = np.array(arr[:])

  inputimg = np.array(inputimgarray, dtype='uint8')
  inputimg = inputimg.reshape((28, 28))

  outputimage = Image.fromarray(inputimg)
  plt.imshow(outputimage,cmap='gray')

□ <matplotlib.image.AxesImage at 0x7fed697f0518>
  0
  5
  10
  15
  20
  25
  0 5 10 15 20 25

```




Figure 7. Sample Input to Model for testing

Since, the model takes 32*32 size images, the input image needs to be padded with zero.

Now this new input image is passed to the model for generate prediction and result alphabet

number is stored in `imp_pred` variable. As can be seen in the Figure 8, the model gave correct output for the sample image as 'Y'.

Predict the pixel

```
[87] # pixel reshape
      pred_img = inputimg.reshape(1,28, 28, 1)
      #pad it to make 32 32
      pred_img = np.pad(pred_img, ((0,0),(2,2),(2,2),(0,0)), 'constant')
      print(pred_img.shape)
      imp_pred = model.predict(pred_img)
```

```
[88] # print(imp_pred)
      letter = np.argmax(imp_pred, axis = 1)
      letter = letter.reshape([len(imp_pred),1])
      print('Predicted ASL Alphabet:',chr(65+letter))
```

Predicted ASL Alphabet: Y



Figure 8. Prediction output on sample image

III. DELIVERABLE 2 : SIGN LANGUAGE GESTURE CREATION IN A SHOP SETTING USING ANIMATIONS IN UNITY

We created a Unity Animation on humanoid avatars that does a sign language gesture as part of Deliverable 2. This small clip of animation is used to apply on humanoid avatar placed in shop setting of the scene in Unity. Finally, the end output of this deliverable is an mp4 video that captures the humanoid avatar doing some ASL signs. In this semester, we choose to do this deliverable for an okay sign. These animations could be created for any given gesture by following the steps detailed further in this section. This deliverable would enable us to generate a synthetic dataset for any gesture that we want the end system to be capable of predicting. So, this deliverable adds extensibility for the writing project in next semester. The final animation clip is to train our end system for American Sign Language detection from video feeds.



Figure 9. Okay Sign by Humanoid Avatar

In this Deliverable 2, the animation was built to make the humanoid avatar do an Okay Sign. For this first, we did rigs configuration for the avatar and then configured the Dopesheets. 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 timeline view has two modes, Dopesheets, and Curves. In Unity, Dopesheets allows viewing each property's keyframe sequence in an individual horizontal track as shown in Fig 10. The Animation Curve means that the Animation Clip controls how a property changes over time shown in Fig 11.

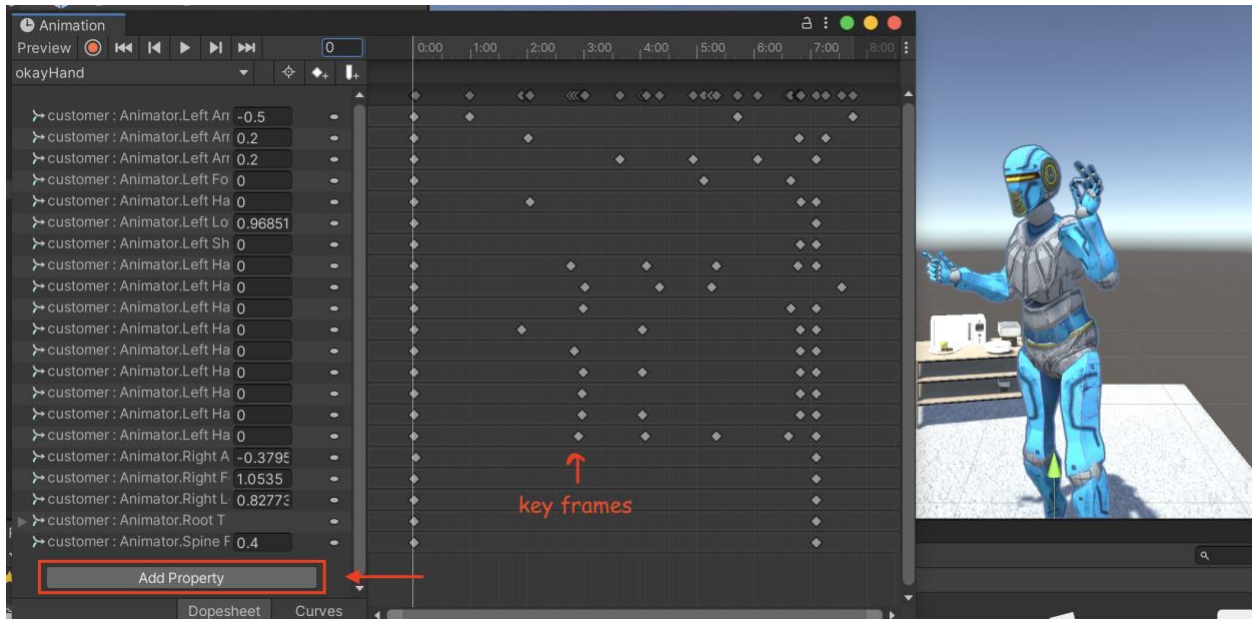


Figure 10. Dopesheet mode in Animation Window in Unity

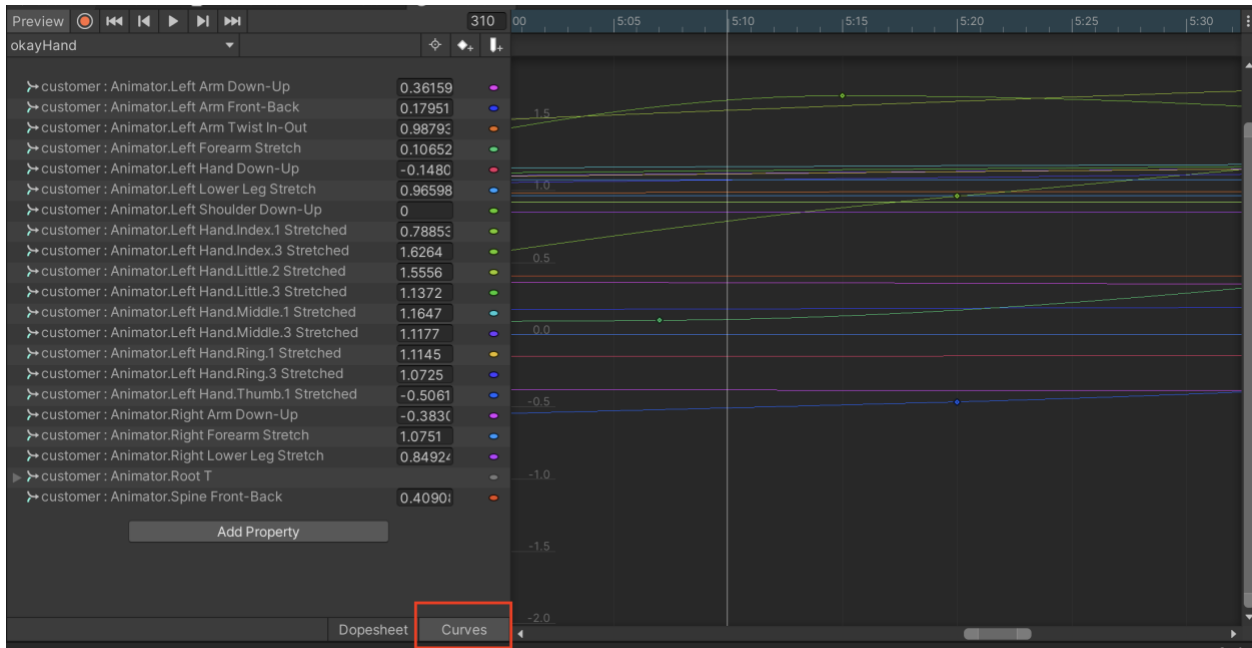


Figure 11. Curves mode in Animation Window in Unity

For this deliverable, all the animations are created from scratch by configuring each

bones points that are required to create a gesture by clicking on ‘Add Property’ in the Animation window. This opens a list of available animators that provides the in-built movement for the game object as shown in Fig 12. So, a combination of these animators is used to make an avatar perform specific gestures.

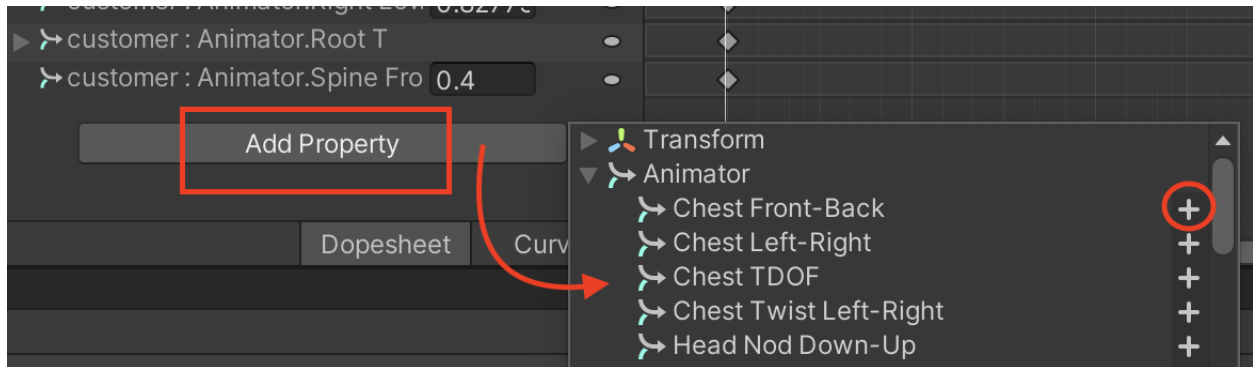


Figure 12. Add Property in Animation

In a given time frame, each key point in the Dopesheets at each point of time determines the state of that bone. It tells the action performed by the model at that time. So, once all the required bones are added as properties to the animation, then movements for each of these bone points at each point of time in the keyframe was configured in the Dopesheet mode as shown in Fig 10.

Lastly, now this animation needs to be added as a state in Unity Animator to be attached to the game object as shown in Fig 13. The final scene in Unity was created to give a shop’s look and feel as shown in Fig 14.

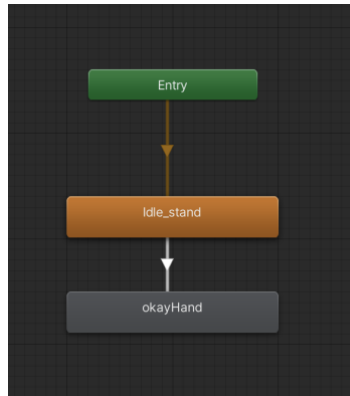


Figure 13. Okay Sign Animator



Figure 14. Final Unity Scene in Shop Setting

IV. DELIVERABLE 3 : HUMANOID GESTURE SYNTHETIC DATASET CREATION IN UNITY

The third deliverable was to generate a video dataset capturing gestures done by a humanoid avatar from different angles. Since the training of the computer vision model needs large data sets, so we choose to create 50 videos for each gesture. This deliverable was specifically aimed at learning to create large datasets as per our requirements. In this semester, as the first step towards video creation for all sign language hand gestures that we would do as part

of the writing project in next semester, we used the animation clips created in Unity that does an okay sign gesture in Deliverable 2 as an input dataset for Deliverable 3. The output dataset would then be used for training the AI model that is capable of recognizing the actions done in sign language by the humanoid avatar.

To get a quality dataset and maintain heterogeneity in each video, each of the videos in the dataset was created by recording it from various angles in the Unity scene. In our writing project next semester, we would have to generate a very large dataset for each kind of gesture. So, manually setting the camera in Unity to record from different angles was not a feasible approach. So, we used C# scripting in Unity to position the camera randomly in 3D space.

In Unity, the C# scripts are usually created within it directly that can be attached to GameObjects [7]. So, here we attached this script to the main camera in Unity. The 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 15. 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 15. 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 Figure 16.

```
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 16. C# code Snippet 2

So, this script was used to generate random location for the camera object and finally it recorded the video in mp4 format using Unity Recorder as shown in Fig 117.

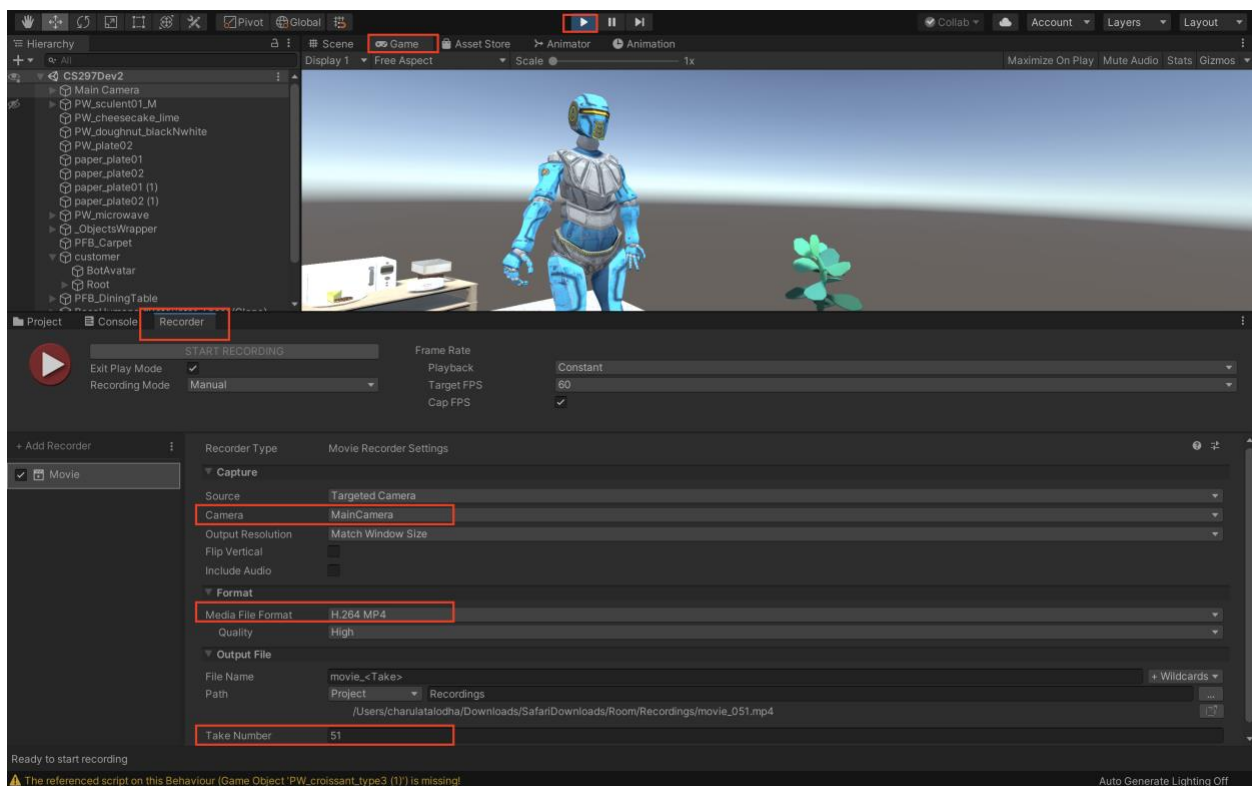


Figure 17. Unity Recorder

V. DELIVERABLE 4 : BONE KEY POINTS DETECTION USING OPENPOSE AND STUDYING SKELETON-BASED ACTION RECOGNITION

The fourth deliverable has two parts first to generate bone points dataset and secondly to study a paper on action recognition using these bone point co-ordinate information. After studying various models, we decide to use the OpenPose model [8] to generate a video dataset that shows bone points for humanoid avatar videos that were submitted as part of Deliverable 3. And for part2, Skeleton-based action recognition with convolutional neural networks [9] paper was studied.

OpenPose has represented the first real-time multi-person system to jointly detect various human parts like body, hand, facial, and foot key points (in total 135 key points) on single images. The core features of OpenPose that are importantly used for this deliverable is 3D real-time single-person keypoint detection. It provides features like 3D triangulation from multiple single views, synchronization of flir cameras handled and compatible with Flir/Point Grey cameras.

Current state-of-the-art approaches to skeleton-based action recognition are mostly based on recurrent neural networks (RNN). In this paper, Chao Li and et al proposed a novel convolutional neural networks (CNN) based framework for both action classification and detection. Raw skeleton coordinates as well as skeleton motion are fed directly into CNN for label prediction. A novel skeleton transformer module is designed to rearrange and select important skeleton joints automatically as shown in Fig. 20

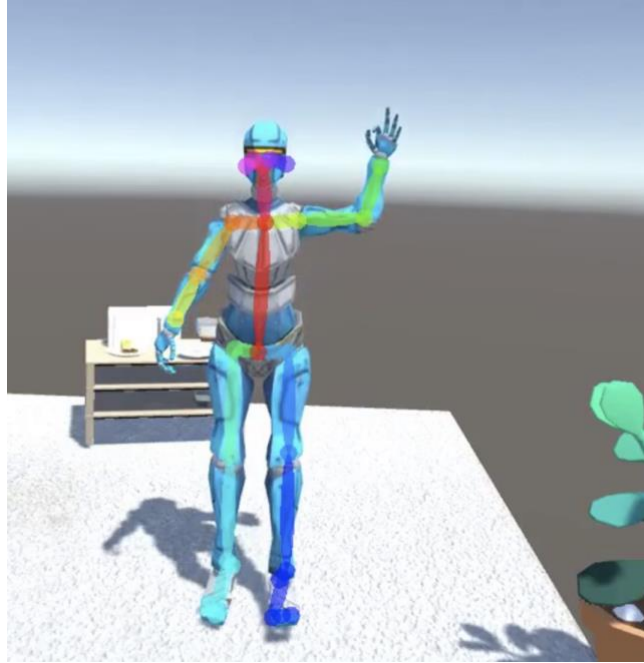


Figure 18. Sample output video from OpenPose

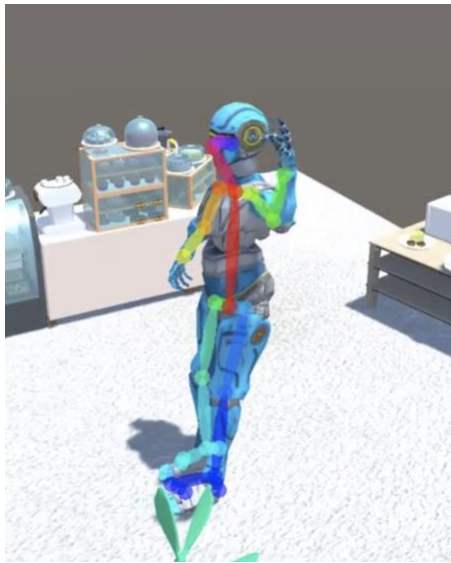


Figure 19. Sample output video from OpenPose

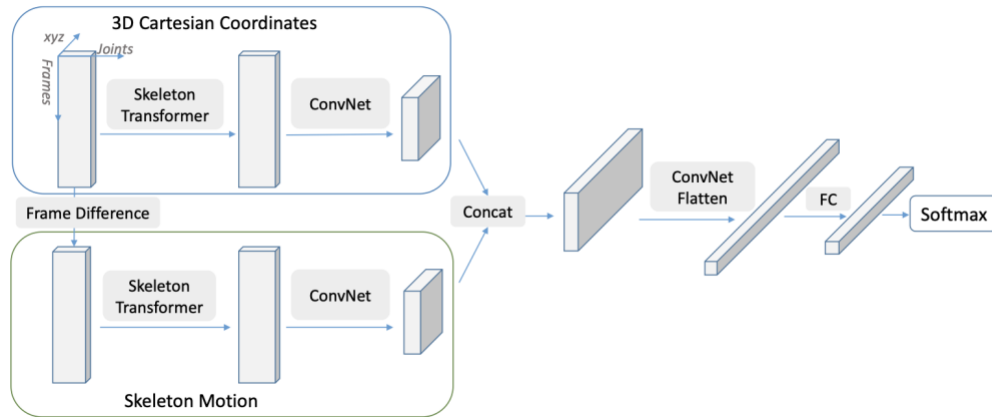


Figure 20. Sample output video from OpenPose

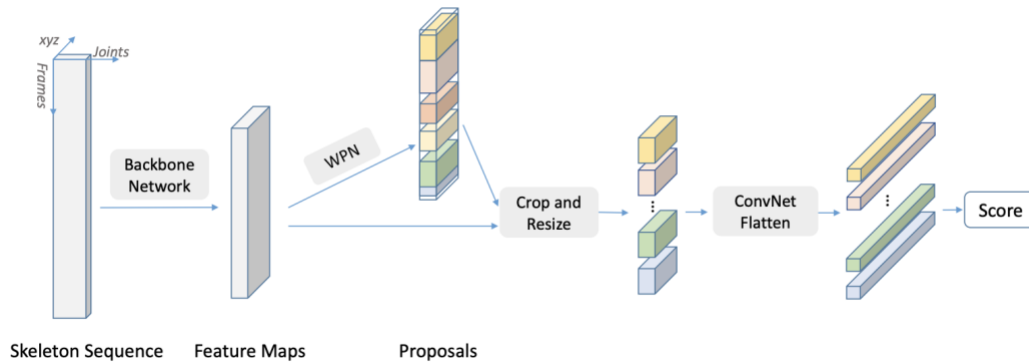


Figure 21. CNN representation of skeleton sequences for action classification.

As displayed in Fig.21, instead of region proposal network (RPN) it used window proposal network (WPN). The 2D anchors are flattened to 1D anchors. Window proposals along the temporal dimension are extracted based on pre-defined anchors. In this architecture, window regression instead of bounding box regression is performed to refine the temporal position of generated window proposals. It then pools the features of each window from the shared feature maps with the crop-and-resize operation once proposals are ready. These features are then fed to the R-CNN subnetwork for classification and window regression.

VI. CONCLUSION

During this semester, as a part of the writing project this semester, we explored various Convolution Neural Networks architectures, Unity animations, and C# scripting that will aid in the development of our final system as part of the writing project in next semester. In this semester, in first deliverable, we studied about LeNet architecture and then implemented it to build a model capable of recognizing the static ASL alphabets. Next, we explored Unity to learn and create custom animations on humanoid characters. This deliverable will help writing project next semester for the creation custom dataset having various ASL gestures.

After a small animation clip for the okay sign gesture was ready, we worked on creating shop setting scenes in Unity. Further, as part of Deliverable 3, these clips were recorded from various angles to generate the synthetic custom dataset required for training the AI model. In next semester, this approach would be used to generate a dataset for varied signs. These videos were lastly 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 3D coordinates for key points like hand, legs, face and others.

Lastly, a paper on ‘Skeleton-Based Action Recognition with Convolutional Neural Networks’ was studied which gave insights into how we can use the 3D coordinates and recognize the gestures done by the humanoid avatar. All these deliverables have built up a strong foundation for our writing project in next semester.

In the next semester, we will try to create animation clips and corresponding datasets for at least five different ASL keywords using the methodology proposed in Deliverable 2 and 3. Also, we will work on improving the accuracy of the ASL alphabets detection model by training it on a custom dataset. We will try to implement the Skeleton-Based Action Recognition model for all the signs supported by our system. Lastly, we will deploy it on some device that can continuously provide video feed and recognize the ASL and finally give out its corresponding English text.

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. 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]
4. “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].
5. “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].
6. 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].
7. Unity Technologies, “Creating and Using Scripts,” *Unity*. [Online]. Available: <https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html>. [Accessed: 15-Dec-2020].
8. CMU-Perceptual-Computing-Lab, “CMU-Perceptual-Computing-Lab/openpose,” *GitHub*. [Online]. Available: <https://github.com/CMU-Perceptual-Computing-Lab/openpose>. [Accessed: 15-Dec-2020].
9. 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].