Detecting and Predicting Visual Affordance of objects in a given environment

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

Bhumika Kaur Matharu

December, 2020

ABSTRACT

Affordance specifies the functionality allowed by an object to a user (or agent) i.e., what actions user can perform on a given object in the environment. Robots take advantage of the concept of affordances to perform human-like actions in the real world. In this research project, the main aim is to create a model to detect and predict visual affordance of an object in an unknown environment through neural networks and deep learning. The focus of the project is to perform visual affordance learning on the synthetic video dataset generated using Unity Gaming Engine.

In this semester, we divided the initial phase of the project into four deliverables. Initially, we implemented multi-class object classification and detection on self-generated emoji dataset using Keras and OpenCV. Later on, we worked towards the generation of synthetic video dataset using Unity. After the creation of the dataset, we researched the existing models being used to learn object affordance from videos. On researching existing methodologies, we chose and tried to implement one of the models discussed to form a baseline model for the project.

***Index Terms* – Convolutional Neural Networks (CNNs), Visual Affordance, Unity**

I. INTRODUCTION

Today, many companies like Amazon, Apple, Walmart, Facebook, Google and Microsoft are taking advantage of machine learning to raise the bar for customer expectations, especially in the retail sector [1]. With the help of computer vision techniques and video analytics, companies are trying to efficiently manage grocery stores by automating most of the manual or repetitive tasks. Automated tasks like cashier-less stores, shelf monitoring, store surveillance, etc., help in achieving better shopping experience for the user as well as the employer. In this project, we propose to utilize computer vision and deep learning models on videos of grocery stores to detect and predict the affordances of objects. Affordance defines the relationship between an object and the actions a user can execute in any given environment. Through affordance detection, we can do shelf monitoring by checking the items available on aisles of the stores and re-stocking them in time. Also, we can create a cashier-less shopping service for the user.

In 2018, Amazon opened Amazon Go shops to provide cashier-less experience to users. In Amazon Go, items taken off the shelf are automatically detected and tracked. After the user completes shopping, they can leave the sore without checking-out as the charge will be deducted from the user's account automatically. Despite potential security threats, Amazon Go is expanding to many other locations in United States. In 2014, a similar technology Scan and Go was launched by Walmart where user can scan and pay for the items purchased via phone. Due to low participation and poor feedback, this technology was suspended in 2018.

In the project, the main focus is to gather an understanding of the user's activity with the items present in the given environment through visual affordance. Most of the research conducted has used RGB images or real-life video recordings of people performing a set of actions. For this project, we generate synthetic video dataset generated using Unity Gaming Engine.

The rest of the report is organized as follows: In Section II, we perform multi-class object classification and detection on self-generated emoji dataset to gain knowledge on how deep learning neural network works. Here, we learn how to manipulate self-generated dataset to improve performance of the model. In Section III, we work towards the generation of synthetic video dataset using Unity Gaming Engine. A total of 50 videos were generated for initial phase of training the model. In Section IV, we discuss existing methodologies implemented to learn affordance of an object. In Section V, we try to implement one of the discussed methodologies to get an understanding and form the baseline model for our project.

II. DELIVERABLE 1: PERFORM OBJECT DETECTION AND IMAGE CLASSIFICATION USING OPENCV AND KERAS

The objective of this deliverable is to obtain basic understanding of how a convolutional neural network works and to get acquainted with other concepts of computer vision. I also learnt how to manipulate the self-generated dataset to improve the performance of the model. For this deliverable, I generated an emoji dataset using the PIL library in Python. For the dataset, five different kinds of emotions are selected. Among those 5 emotions, at most 3 of them were written (drawn) to an image. In the program, random emoji size and position are selected for each image. Using the program, a total of 1000 images are created. This dataset is further divided into training and test set.

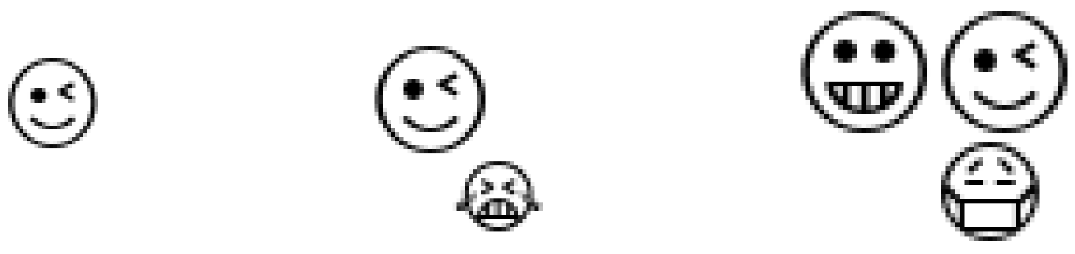


Fig 1. Emoji Dataset

This deliverable is performed in 2 steps. In the first step, I utilized Open CV to detect the number of emojis present in an image. Since the emojis doesn't consist of any human like features, I was not able to implement Haar feature-based cascade classifier to detect emojis. So instead, I used CV2 Hough Circles to detect all the emojis in the image. The total number of emojis detected was saved for final prediction.

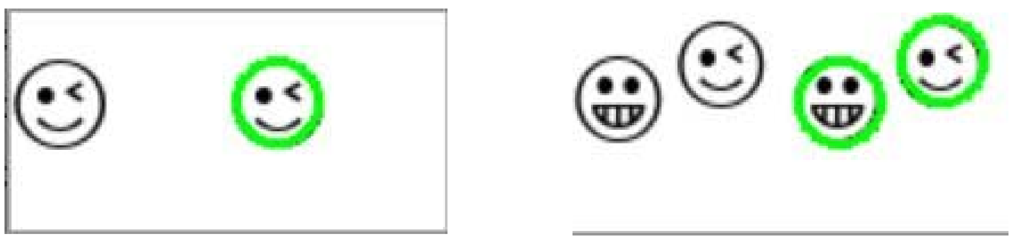
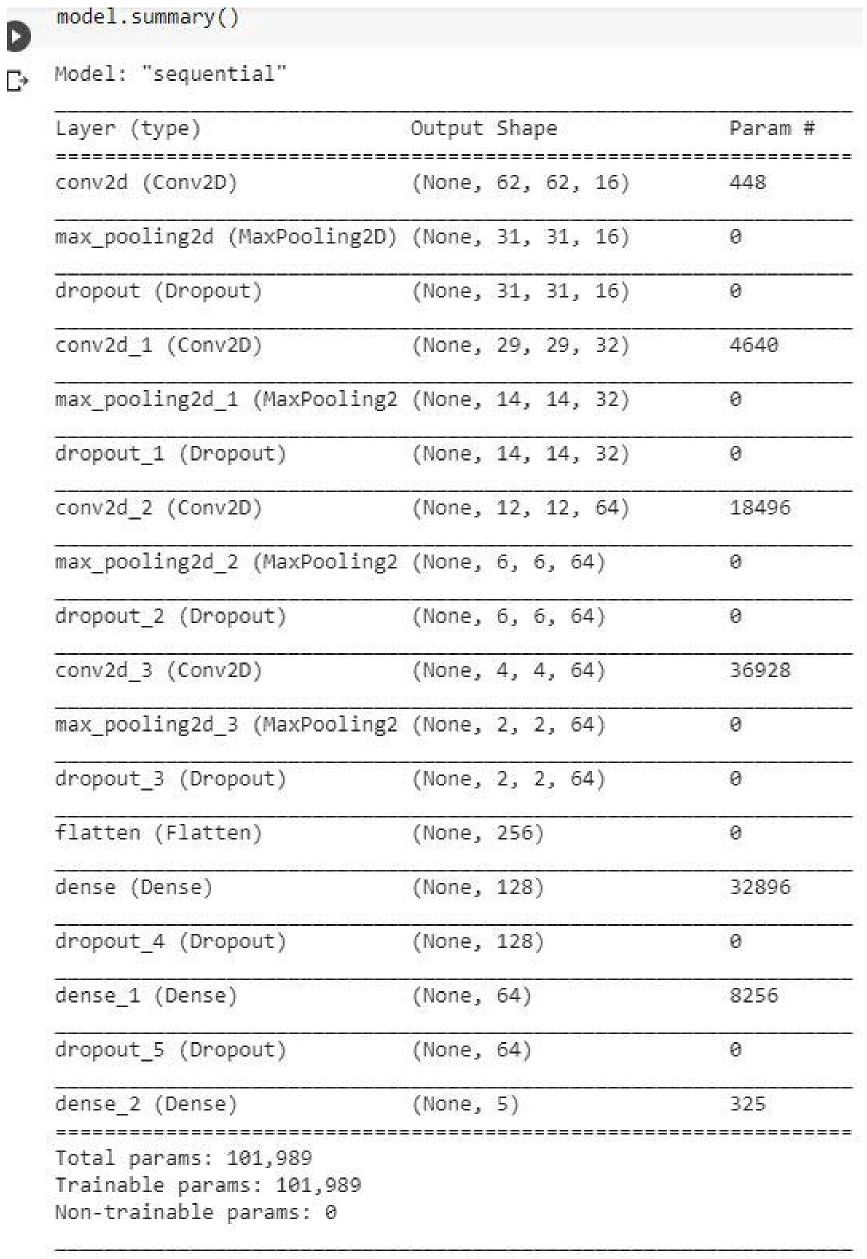


Fig 2. Detecting no. of emojis in Open CV

In the second step of this deliverable, I created a CNN to perform multi-class classification which predicts the label (emotion) for all the kind of emojis present in an image. Along with the labels (emotions) of emojis, the locations of all the emojis present in an image were also given as an input to the model. For example, the emoji with the lowest x coordinate gets the location ‘1’, the emoji with second-lowest x coordinate gets the location ‘2’ and the emoji with farthest x coordinate gets the location ‘3’.

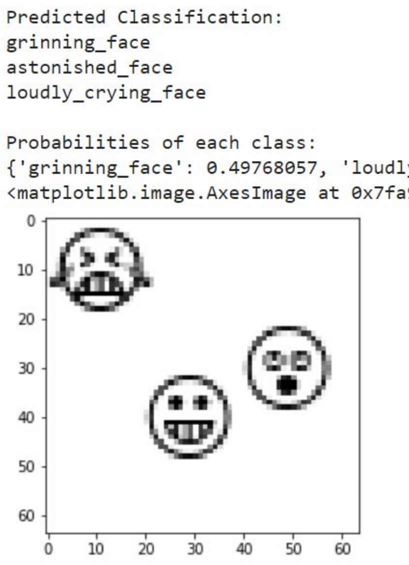


```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 16)        448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 16)        0
_________________________________________________________________
dropout (Dropout)            (None, 31, 31, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 128)               32896
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 325
=================================================================
Total params: 101,989
Trainable params: 101,989
Non-trainable params: 0
_________________________________________________________________
```

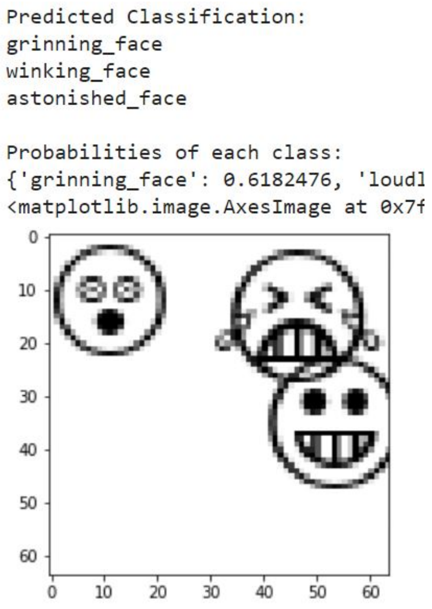
Fig 3. Model Architecture

The accuracy of the model was not up to expectation due to uneven weight distribution amongst each label (emotion). By combining the results of both the steps i.e., the total number of emojis and the prediction label of all emojis, we get top N label predictions with the highest probability. Here N denotes the total number of emojis detected through Open CV.



```
Predicted Classification:
grinning_face
astonished_face
loudly_crying_face

Probabilities of each class:
{'grinning_face': 0.49768057, 'loudly_
<matplotlib.image.AxesImage at 0x7fa9:
```

Fig 4. Correct Classification



```
Predicted Classification:
grinning_face
winking_face
astonished_face

Probabilities of each class:
{'grinning_face': 0.6182476, 'loudl
<matplotlib.image.AxesImage at 0x7f
```

Fig 5. Incorrect Classification

III. Deliverable 2: Generation of Synthetic Dataset Using Unity

The focus of the project is to learn affordances from videos, so the objective of this deliverable is to generate synthetic (virtual) dataset using the Unity Game Engine. In the deliverable, the scope of the dataset was kept small given the time frame. To generate a total of 50 videos, same scene(activity) is recorded in different surroundings and from different camera angles.

In the scene, a humanoid will walk towards a table i.e., target position consisting of many objects and will pick up an object i.e., a book for our scenario.

1. Initially an environment is built by placing (or adding) various Game Objects which include a Humanoid-Player (Ethan), a table consisting of many objects like books, food items etc. in the environment.

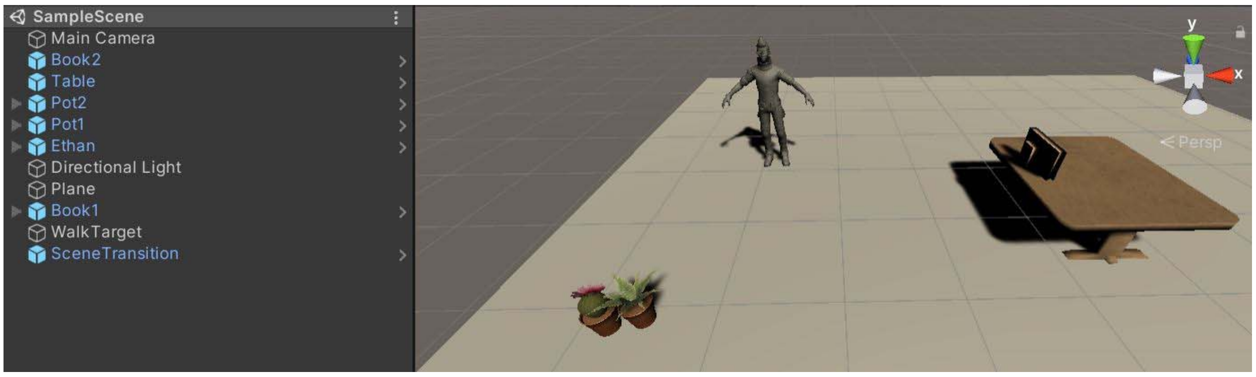


Fig 6. Scene 1

2. After creating an environment, the next task is to move the player to the target position. To control the movement of the player, third-person animator controller of the player is utilized to allocate a target position. The player will walk to the assigned target position in every scene to pick up the objects. To let the player, know that it is near the object a box collider was placed around the object (i.e. book) to be picked up by player. As the player enters the box collider, a

trigger will initiate and enable the player to pick up the requisite object.

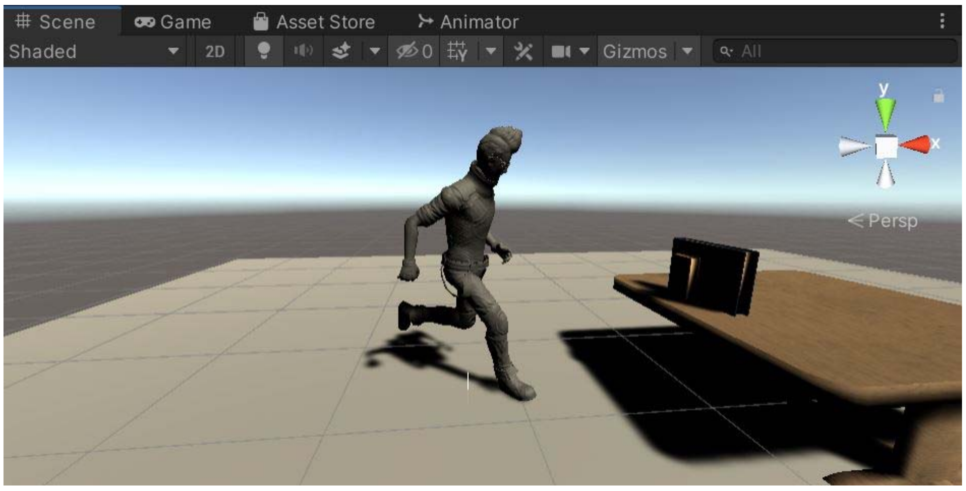


Fig 7. Player walking towards target position

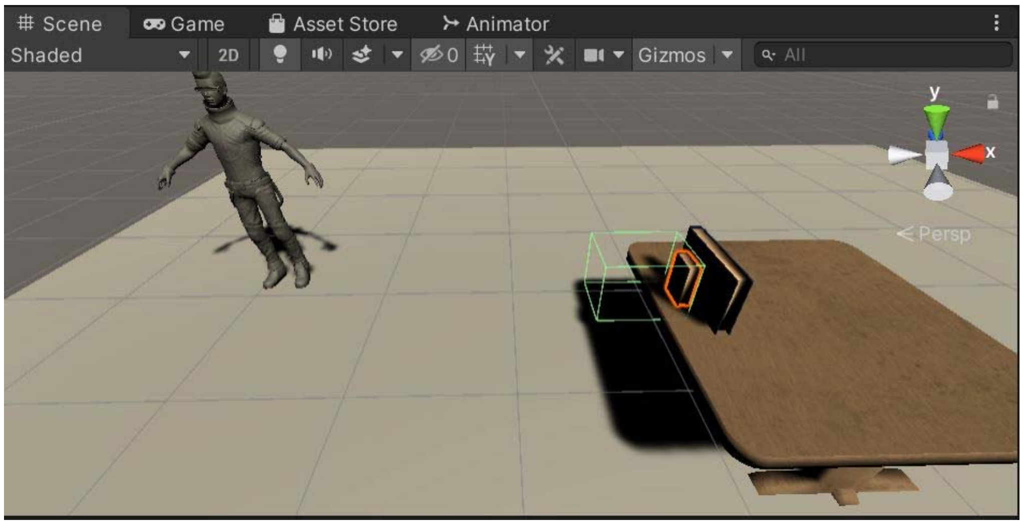
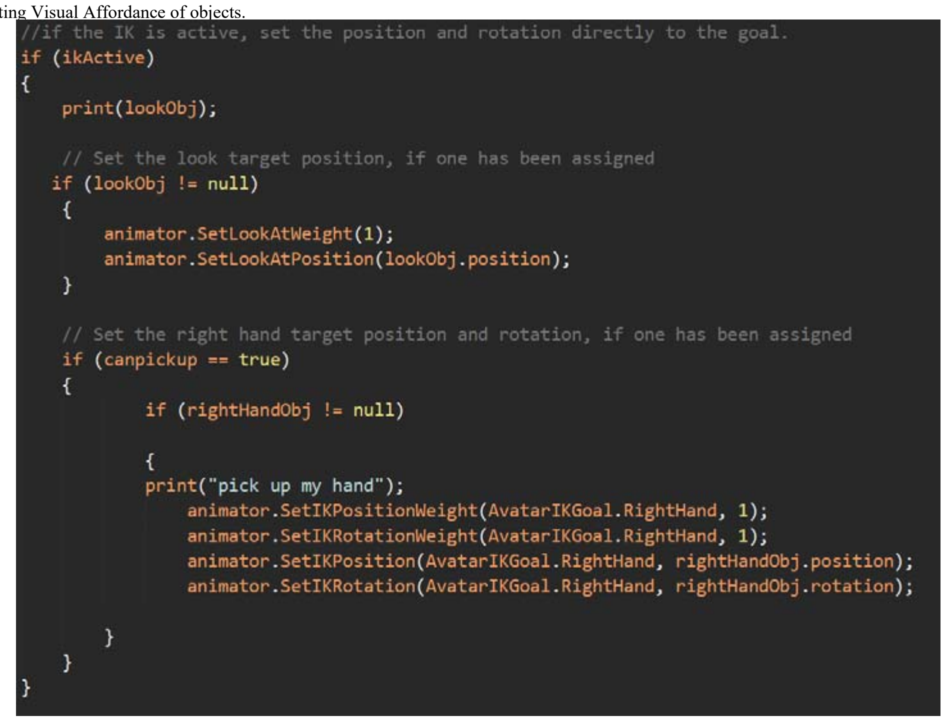


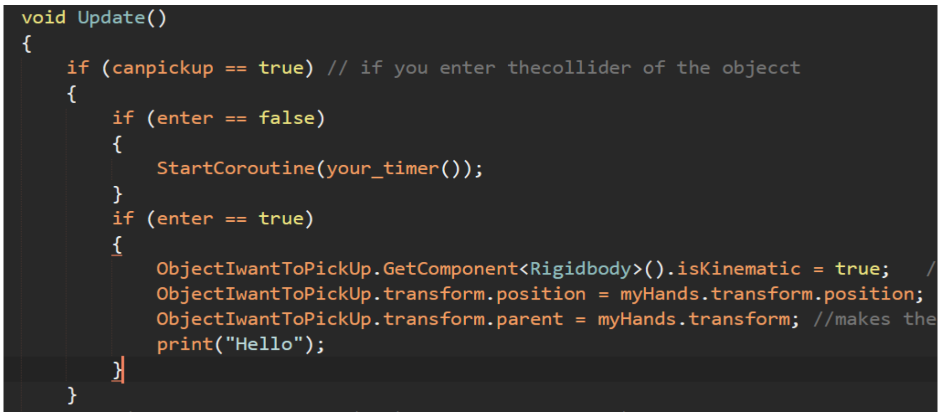
Fig 8. Box collider around the book (Green cube)

3. C# scripts are applied to manipulate the movements of the player to pick up an object. To make the scene a bit more realistic, I took advantage of inverse kinematics to move the arm of the player while picking up an object. Inverse kinematics allows to rotate or move the skeleton of player Game Object to predefined values. On entering the box collider, the player's arm will move towards the object. After a delay of 1 sec, coordinates of the book is set to the coordinates of player's hands. Below is a brief snippet of the C# scripts.



```
//if the IK is active, set the position and rotation directly to the goal.
if (ikActive)
{
    print(lookObj);

    // Set the look target position, if one has been assigned
   if (lookObj != null)
    {
        animator.SetLookAtWeight(1);
        animator.SetLookAtPosition(lookObj.position);
    }

    // Set the right hand target position and rotation, if one has been assigned
    if (canpickup == true)
    {
            if (rightHandObj != null)

            {
            print("pick up my hand");
                animator.SetIKPositionWeight(AvatarIKGoal.RightHand, 1);
                animator.SetIKRotationWeight(AvatarIKGoal.RightHand, 1);
                animator.SetIKPosition(AvatarIKGoal.RightHand, rightHandObj.position);
                animator.SetIKRotation(AvatarIKGoal.RightHand, rightHandObj.rotation);

        }
    }
}
```

Fig 9. Code: Arm Movement – Inverse Kinematics



```
void Update()
{
    if (canpickup == true) // if you enter thecollider of the objecct
    {
        if (enter == false)
        {
            StartCoroutine(your_timer());
        }
        if (enter == true)
        {
            ObjectIwantToPickUp.GetComponent<Rigidbody>().isKinematic = true;   /
            ObjectIwantToPickUp.transform.position = myHands.transform.position;
            ObjectIwantToPickUp.transform.parent = myHands.transform; //makes the
            print("Hello");
        }
    }
```

Fig 10. Code: Picking up an Object

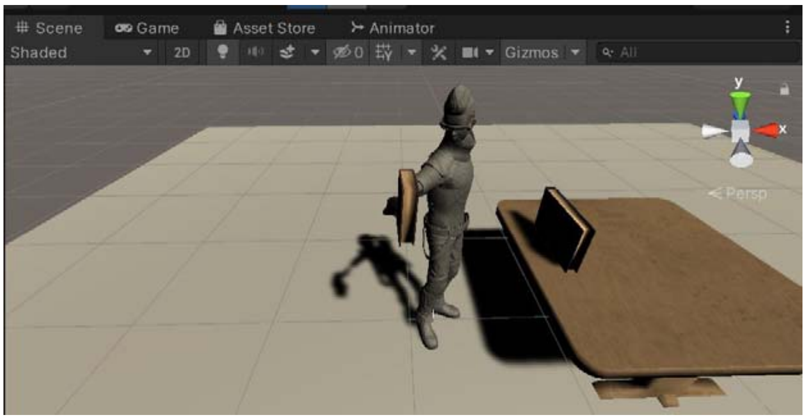
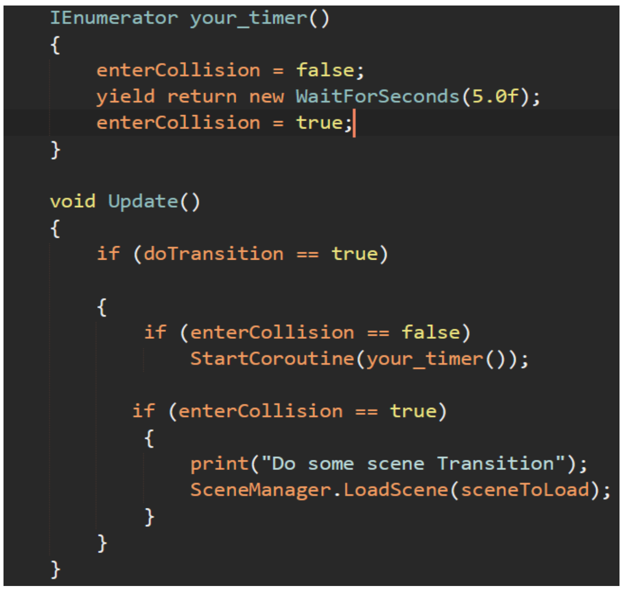


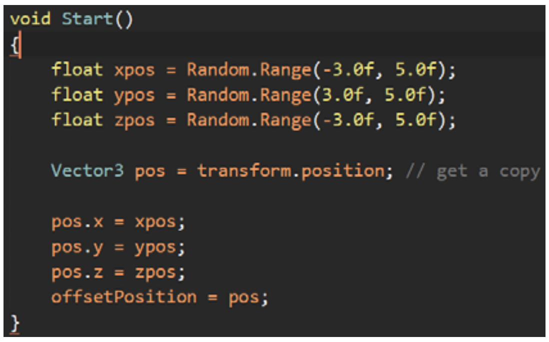
Fig 11. Player picking up an object

4. To create a large number of videos, I recorded the same scene from multiple camera angles. Camera angles coordinates are randomly chosen through C# script. On completing the task of picking up the object, after 5 secs delay, the player will transition into the same scene with different viewpoint (camera angle). After transitioning to the same scene, the player will again perform the same activity and it is recorded from different camera angle. Below is a brief snippet of C# scripts.



```
IEnumerator your_timer()
{
    enterCollision = false;
    yield return new WaitForSeconds(5.0f);
    enterCollision = true;
}

void Update()
{
    if (doTransition == true)

    {
        if (enterCollision == false)
            StartCoroutine(your_timer());

       if (enterCollision == true)
        {
            print("Do some scene Transition");
            SceneManager.LoadScene(sceneToLoad);
        }
    }
}
```

Fig 12. Code: Loading new scene



```
void Start()
{
    float xpos = Random.Range(-3.0f, 5.0f);
    float ypos = Random.Range(3.0f, 5.0f);
    float zpos = Random.Range(-3.0f, 5.0f);

    Vector3 pos = transform.position; // get a copy

    pos.x = xpos;
    pos.y = ypos;
    pos.z = zpos;
    offsetPosition = pos;
}
```

Fig 13. Code: Random Camera position

5. Seven such scenes are created with different surroundings and objects in each scene. These videos are split into individual videos using FFMPEG tool generating a total of 50 videos.

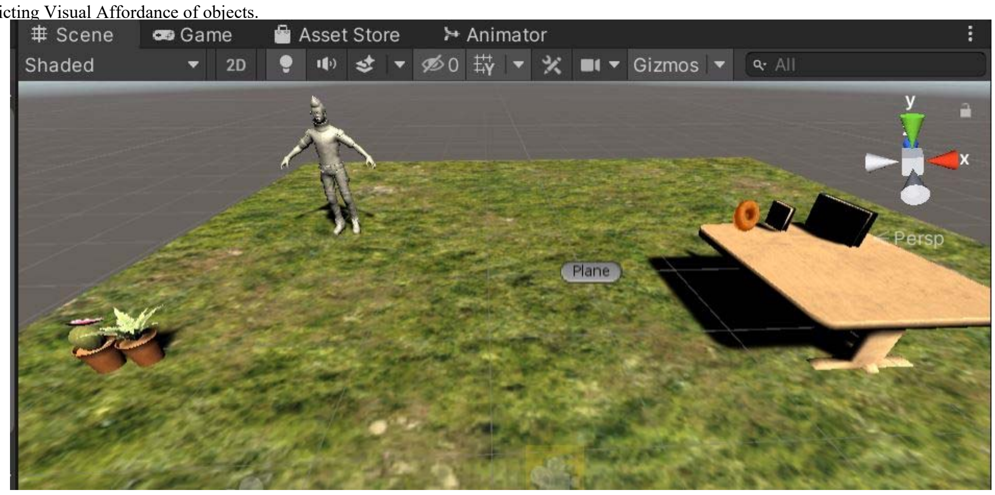


Fig 14. Scene 2

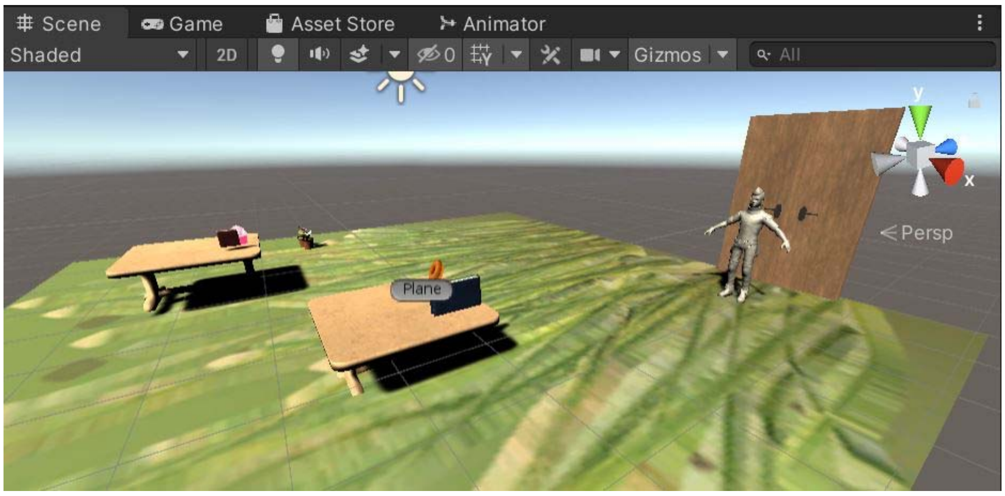


Fig 15. Scene 3

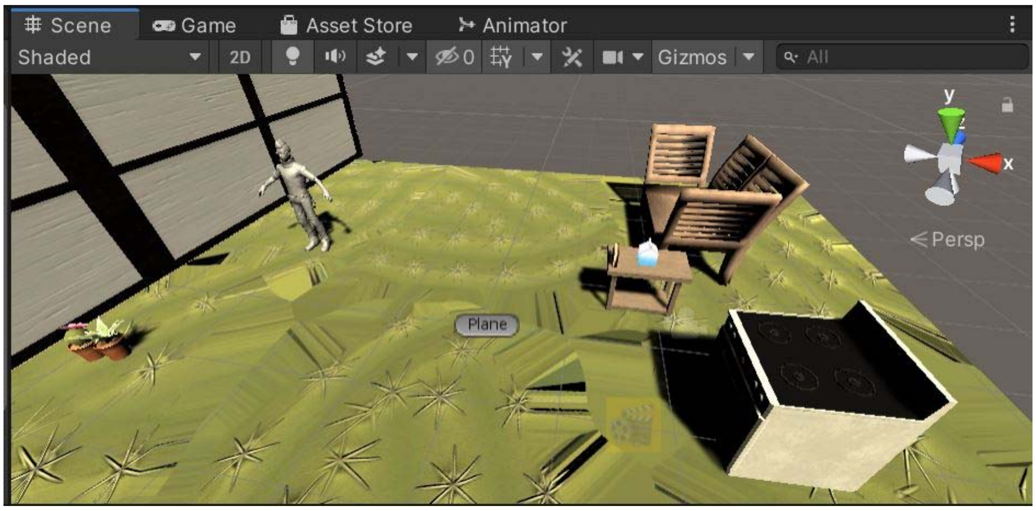


Fig 16. Scene 4

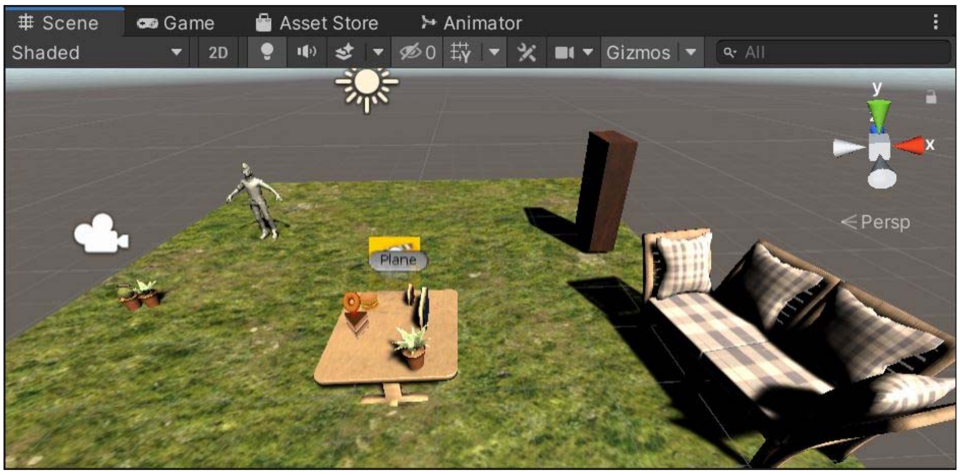


Fig 17. Scene 5

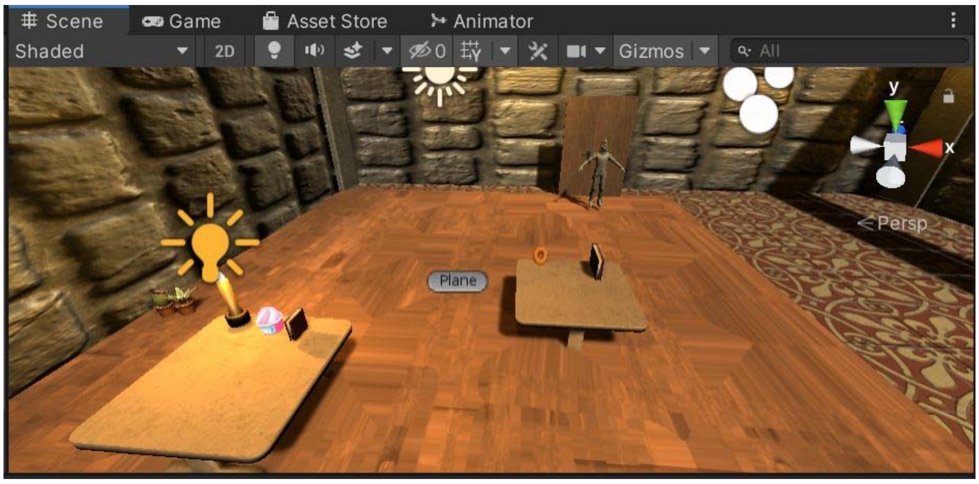


Fig 18. Scene 6

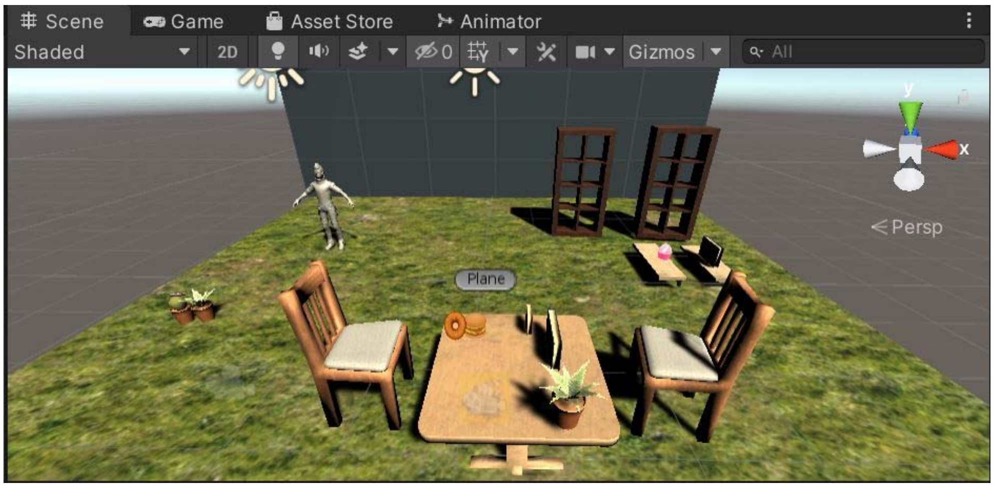


Fig 19. Scene 7

IV. DELIVERABLE 3: EXISTING METHODOLOGIES ON AFFORDANCE

In this deliverable, I researched about the on-going experiments conducted to detect and predict visual affordances from video. I studied [3], [4], [5] to gain understanding about how to capture temporal affordance information from videos. Demo2Vec model discussed in [4] aligned more towards the goal of our project.

In [4] Kuan Fang et al. introduced Online Product Review dataset for affordance (OPRA) which consists of labelled YouTube product review videos of kitchenware objects, household appliances, consumer electronics, tools etc. Each full-length video is split into 2-15 segmented clips where the user is demonstrating some activity on a product. There are a total of seven actions(activity) classes in the dataset - hold, touch, rotate, push, pull, pick up, put down. Along with these videos, up to 5 images of the product are also collected for each video clip. These static images are annotated with ten points highlighting the interaction region between the user and the object.

Kuan Fang et al. developed a model, Demo2Vec, to take advantage of the extracted feature embedded vector from the segmented clips to predict the interaction region and the action class for the object [4]. The Demo2Vec consists of 2 parts, a demonstration encoder and an affordance predictor. The demonstration encoder takes the input of a product (object) image and the segmented clip where the user is demonstrating particular action on the product. It then converts this input to demonstration embedded vector. The demonstration embedded vector is then given as an input to the affordance predictor. The affordance predictor uses the embedded vector to predict the action class and heatmap of the interaction region.

To implement demonstration encoder, two ConvLSTM networks were used. In one of the ConvLSTMs, an RGB image of the product is given as input and in the other, demonstrated motion clip of the product is given as input. The demonstrated clip is subsampled to 5 frames per second and the RGB image is resized to 256*256. These

ConvLSTMs extract the spatial and temporal affordance information of the product and provide the demonstration embedding vector.

Affordance predictor consists of an action classifier and a heatmap decoder. The action classifier implements an LSTM to predict the action class of the product [4]. To generate the heatmap of the interaction region of a product, fully convolutional neural network is utilized which concatenates the convolutional features with demonstration embedding vector. To generate the heatmap around interaction region these concatenated features are given as an input into transpose convolutional layers.

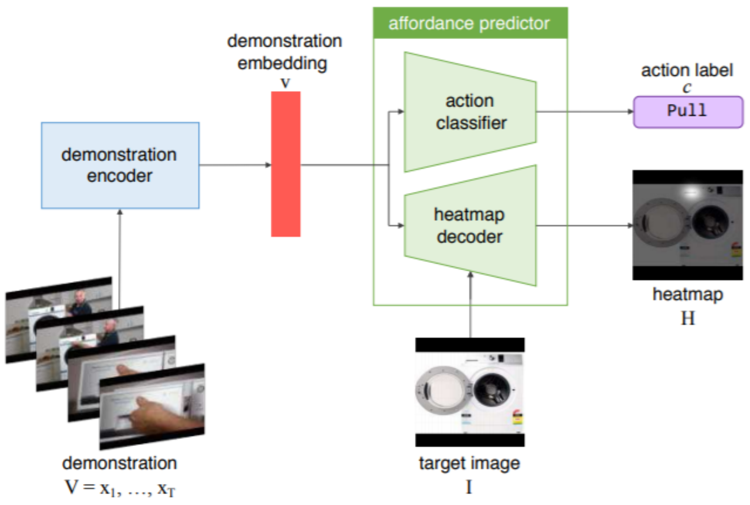


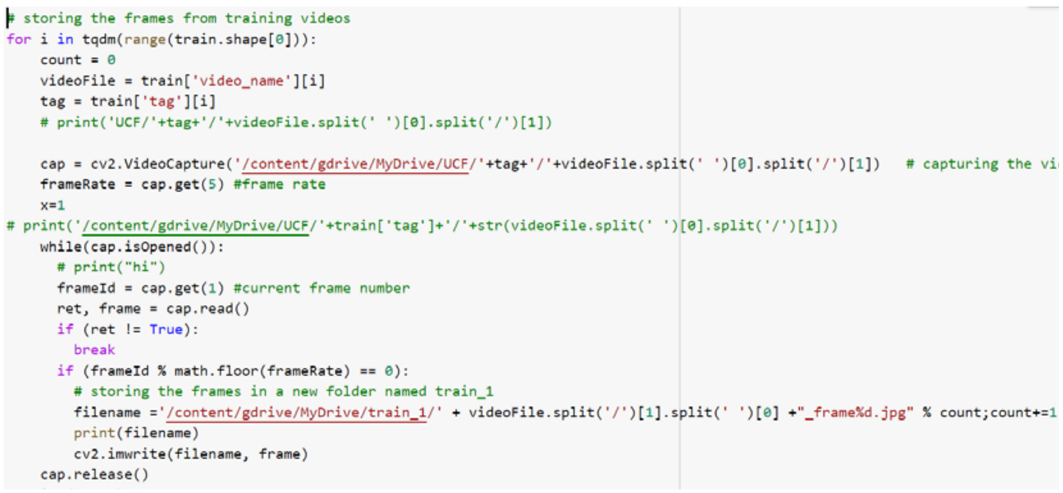
Fig 20. Model Overview [4]

The OPRA dataset introduced by Kuan Fang et al. is known to be more diverse dataset as compared to other datasets in terms of different variety of objects and action classes [4]. The Demo2Vec model was able to predict the action classes of the product and generate the heatmap for the interaction region on the product. The common cause of failure was when two similar actions are performed on an object. For example, often rotation action was mistaken as grasping action or vice-versa.

V. DELIVERABLE 4: IMPLEMENTING DEMO2VEC MODEL

The objective of this deliverable is to re-create Demo2vec model proposed in [4] as this model aligns with the baseline model we require for our project. The main aim of this deliverable is to gain insight on how can we capture temporal information from videos and utilize it on synthetic video dataset generated through Unity. In this deliverable, I created a partial demonstration encoder to extract embedded features out of the videos.

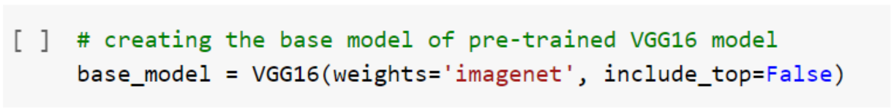
To keep the scope of this model simple, I performed human activity recognition using UCF101 – Action Recognition Dataset as the dataset. To feed video as an input into the neural network, frames were extracted from each video clip at a rate of 5 frames per second. Below is a brief snippet of the code.



```
# storing the frames from training videos
for i in tqdm(range(train.shape[0])):
    count = 0
    videoFile = train['video_name'][i]
    tag = train['tag'][i]
    # print('UCF/'+tag+'/'+videoFile.split(' ')[0].split('/')[1])

    cap = cv2.VideoCapture('/content/gdrive/MyDrive/UCF/'+tag+'/'+videoFile.split(' ')[0].split('/')[1])   # capturing the vi
    frameRate = cap.get(5) #frame rate
    x=1
# print('/content/gdrive/MyDrive/UCF/'+train['tag']+'/'+str(videoFile.split(' ')[0].split('/')[1]))
    while(cap.isOpened()):
      # print("hi")
      frameId = cap.get(1) #current frame number
      ret, frame = cap.read()
      if (ret != True):
        break
      if (frameId % math.floor(frameRate) == 0):
        # storing the frames in a new folder named train_1
        filename ='/content/gdrive/MyDrive/train_1/' + videoFile.split('/')[1].split(' ')[0] +"_frame%d.jpg" % count;count+=1
        print(filename)
        cv2.imwrite(filename, frame)
    cap.release()
```

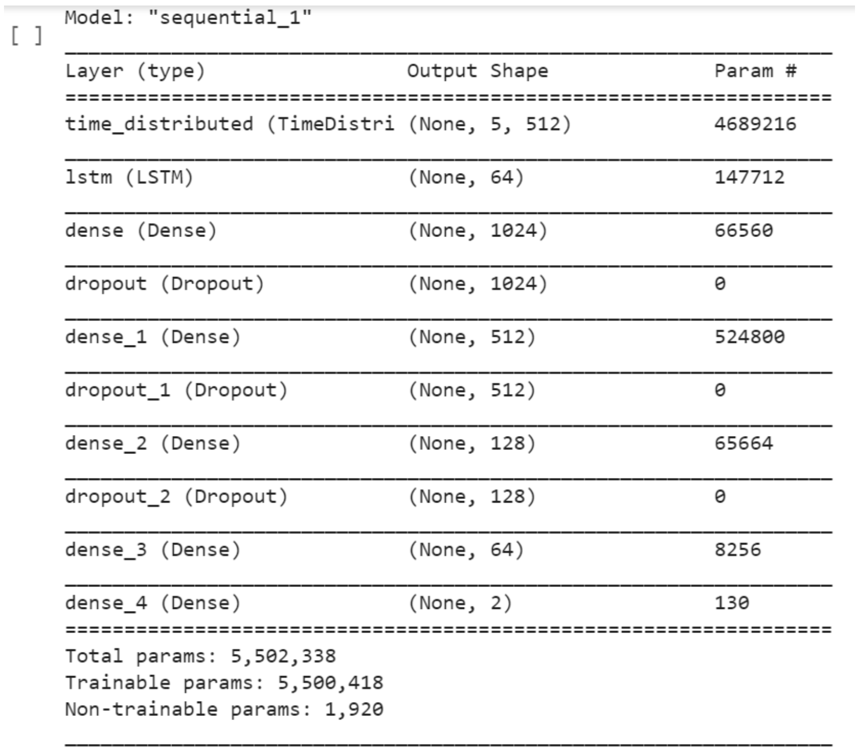
Fig 21. Frame extraction from video clip

I utilized VVG-16 model with pretrained weights restored from imageNet dataset as our base model. For videos, I implemented a Convolutional LSTM network which can handle a time series data and can extract spatial – temporal information from the input. All the collected frames are resized to 112X112. The set of frames extracted from each video clip is first passed through the base model and then fed into the ConvLSTM. The ConvLSTM uses 3 kernel size and 1 stride similar to the Demo2Vec Model [4]. I take advantage of Time Distributed layer on top LSTM layer to provide a temporal layer to each frame fed into the input.



```
[ ]  # creating the base model of pre-trained VGG16 model
     base_model = VGG16(weights='imagenet', include_top=False)
```

Fig 22. Base Model



```
[ ]  Model: "sequential_1"
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #   
     =================================================================
     time_distributed (TimeDistri (None, 5, 512)            4689216   
     _________________________________________________________________
     lstm (LSTM)                  (None, 64)                147712    
     _________________________________________________________________
     dense (Dense)                (None, 1024)              66560     
     _________________________________________________________________
     dropout (Dropout)            (None, 1024)              0         
     _________________________________________________________________
     dense_1 (Dense)              (None, 512)               524800    
     _________________________________________________________________
     dropout_1 (Dropout)          (None, 512)               0         
     _________________________________________________________________
     dense_2 (Dense)              (None, 128)               65664     
     _________________________________________________________________
     dropout_2 (Dropout)          (None, 128)               0         
     _________________________________________________________________
     dense_3 (Dense)              (None, 64)                8256      
     _________________________________________________________________
     dense_4 (Dense)              (None, 2)                 130       
     =================================================================
     Total params: 5,502,338
     Trainable params: 5,500,418
     Non-trainable params: 1,920
     _________________________________________________________________
```

Fig 23. ConvLSTM Model Summary

For OPRA dataset, segmented clips will be extracted using FFMPEG tool. From these segmented clips we will generate frames at a rate of 5 frames per second. In this semester I understood and developed a ConvLSTM network to take video as its input, extract features from the video and predict activity happening in a static image. The rest of the implementation of Demo2Vec will be continued in the next semester.

VI. Conclusion

In this semester, we focus on generating the dataset and trying to develop a baseline model to detect and predict visual affordance. We begin by performing a small multi-class classification on generated emoji dataset. This deliverable gave an insight into how we can manipulate the generated dataset accordingly to get better accuracy. To make a dataset of 50 videos, we created multiple videos performing the same activity in different surroundings. After the generation of synthetic video dataset using the Unity Game Engine, we researched and studied the existing methodologies in the domain of visual affordance. On finalizing an approach, we tried to recreate the proposed model in the approach [4] on OPRA dataset.

In the next semester, we are going to focus on the implementation of the baseline model. First, we will re-create the whole model and perform affordance learning on real-life video dataset i.e., OPRA. Then we will utilize the re-created model, Demo2Vec, to evaluate the performance of the model on synthetic video dataset generated through Unity. We will work towards improving the performance of a model of synthetic video dataset depending on the results achieved. For the initial phase, we kept the scope of the dataset to 50 videos only where the user is picking up the same item. In future, we will add more scenes resembling grocery store environment performing multiple actions in multiple settings to enhance affordance learning of the model.

REFERENCES

[1] V. Kaur, V. Khullar and N. Verma, “Review of Artificial Intelligence with retailing sector” in *Journal of Computer Science Research*, 2020.

[2] S. Harwood, N. Hafezieh, “Affordance - what does this mean?” in *22nd UKAIS Annual Conference,* Oxford, UK, 2017.

[3] M. Hassanin, S. Khan and M. Tahtali, “Visual Affordance and Function Understanding: A Survey” in *Computer Vision and Pattern Recognition,* Utah, USA, 2018.

[4] K. Fang, T. Wu, D. Yang, S. Savarese,J. Lim, “Demo2Vec: Reasoning Object Affordances From Online Videos” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* USA, 2018, pp. 2139-2147

[5] T. Nagarajan, C. Feichtenhofer, K. Grauman, “Grounded Human-Object Interaction Hotspots from Video” in *CoRR*, USA, 2019.