Detecting and predicting visual affordance of objects in a given environment

A Project Report

Presented to

Dr. Chris Pollett

Dr. Robert Chun

Sunhera Paul

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the

Class CS 298

By

Bhumika Kaur Matharu

May, 2021

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my advisor Dr. Chris Pollett, for his guidance, enthusiasm, support, collaboration, and constant motivation throughout the project. All the learnings I achieved in the past year would not have been possible without him. I am incredibly grateful to have an opportunity to work with someone as qualified and as brilliant as him.

I would also like to express my sincerest gratitude to my committee members Dr. Robert Chun and Sunhera Paul, for their time, guidance, and valuable feedback.

Finally, I would like to express my indebtedness to my family and friends for their countless hours of support and motivation over the years.

ABSTRACT

The rapid growth of the development of autonomous robots is transforming the manufacturing and healthcare industry in many ways, but they still face many challenges. One of the challenges experienced by autonomous robots is their inability to manipulate an unknown object without human supervision. One way through which autonomous robots can manipulate an unknown object is affordance learning [1]. Affordance describes the action a user can perform on the object in given surroundings. This report describes our proposed model to detect and predict the affordance of an object from videos by leveraging the spatial-temporal feature extraction through ConvLSTM and Fully Convolutional Networks. Our model is built upon an Encoder-Decoder architecture. The encoder consists of CNN to capture spatial features of the input frames and ConvLSTM to capture the temporal dynamics of the input frames. The decoder utilizes the encoder's output to classify the affordance of a given task and predict the interaction region between the human and the object in the form of a heatmap. The decoder is composed of a LSTM, utilized to classify affordance of a given task, and a Fully Convolutional Neural Network to predict the heatmap of the interaction region.

LIST OF FIGURES

Figure 26 to Figure 67 consists of results and observations obtained after implementing the discussed experiments.

Table of Contents

I. INTRODUCTION

Many tech companies like Apple, Walmart, Amazon, and Google are utilizing the recent advancement in artificial intelligence and computer vision techniques to build autonomous robots. Though autonomous robots provide various advantages to the customer and the employer, it fails to perform automated tasks in the presence of novel objects. Through affordance learning, autonomous robots can adapt and learn how to interact with an unknown object in given surroundings. Affordance specifies the set of actions a user can perform on an object in the given environment. This project proposes to implement a model to detect and predict affordances of a given object along with the interaction region between human and the object.

Various types of research have been conducted in the domain of Affordance on RGB images and RGB videos of human-object interactions. Anh, [2] and Ian, [3] performed multiple object detection along with their respective affordances on RGB images by utilizing deep convolutional networks. Darwin, [4] implemented an encoder-decoder architecture consisting of deep convolutional networks and predicted affordances of various objects as heatmaps. [2] and [4] also demonstrated the effectiveness of their results by applying the methodologies on a real-life humanoid robot (WALK-MAN)

Few RGB images [5] approaches implemented pixel-wise affordance segmentation using convolutional neural networks. Johann, [6] performed pixel-wise affordance segmentation through weaker supervised methods which focus on key point annotations and image annotations. Other approaches [7] utilized human pose estimation to predict the affordances of the objects through human-object interactions.

Hema, [8] carried out a structural support vector machine (SSVM) approach to learn object affordances and the human activity performed in RGB-D videos. They demonstrated the use of the approach on a humanoid robot (PR2 robot). Tianmin, [9] captured the spatiotemporal features of

RGB-D videos and performed weakly supervised methods to transfer the grammar of affordance to humanoid robots to enable human-like interaction. Similarly, Tushar, [10] implemented a weakly supervised approach using LSTM to learn the affordance of the objects and the human-object interaction regions. Kuan, [11] built a strongly supervised encoder-decoder architecture using ConvLSTM to predict the affordance of various objects and human-object interaction as heatmaps on videos.

In this project, an encoder-decoder architecture is developed to perform affordance classification of objects and predict human-object interaction as heatmaps on a video dataset. OPRA (Online Product Reviews for Affordances) video dataset introduced by [11] is used. The encoder consists of a convolutional neural network (CNN) and a recurrent neural network (RNN). A pre-trained convolutional neural network is utilized to capture the spatial features of the video frames, whereas to capture the temporal features of the video, a recurrent neural network has been used. The combined output of these two models produces an encoded spatiotemporal feature embedding of the video. The encoder's output is further utilized to classify affordance and predict interaction heatmap. To classify the object's affordance present in the video, another recurrent neural network is utilized to detect the affordance of a given object. To compute heatmap, a fully convolutional neural network (FCN) is used.

The rest of the report is organized as follows: the background chapter provides information about the techniques and deep learning models leveraged. This is followed by the design and implementation chapters, giving in-depth information about the dataset, pre-processing methodologies, deep learning models implemented, environment, and other hardware requirements. In the experiments chapter, we visualize the results produced by the encoder-decoder architecture implemented. The conclusion and future work chapter concludes the findings of this project and proposes possibilities of future work.

II. BACKGROUND

This chapter describes the technical background of the computer vision techniques and the deep learning models utilized to achieve affordance learning. The first section covers the description of the convolutional neural network along with the pre-trained CNN-VGG16 model leveraged in this project. The following section describes the recurrent neural network used to capture temporal features of the video. The third section illustrates how the spatial features and temporal features are extracted individually. This is followed by the fourth section, which gives detailed information about the fully convolutional network and deconvolution methods used for heatmap prediction.

2.1 Convolutional Neural Networks (CNN)

Most computer vision algorithms use convolutional neural networks (CNN) to analyze and extract features from any kind of image or video data. The use of CNNs is not restricted to computer vision tasks only as it is used for text classification in natural language processing (NLP). We utilize the VGG16-CNN in the project, which extracts features of the visual input.

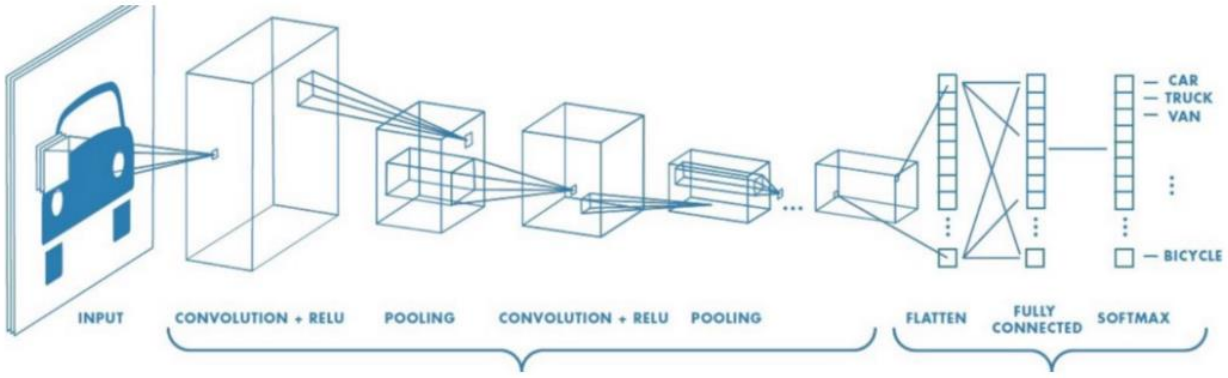

Figure 1. Architecture of CNN [12]

A basic CNN architecture consists of many layers like convolutional layers, activation layers, max-pooling layers and fully convoluted layers. All these layers allow the network to extract features

or information like text, depth, etc., from the input image. The convolutional layer takes a matrix as the input, which represents the input image. This layer performs convolution operation by iterating over the matrix in a small matrix and extract different features in the form of feature maps. To bring non-linearity to the extracted data, an activation function is used. There are different activation functions like ReLU, Tanh, Leaky ReLU, all serving different functionality.

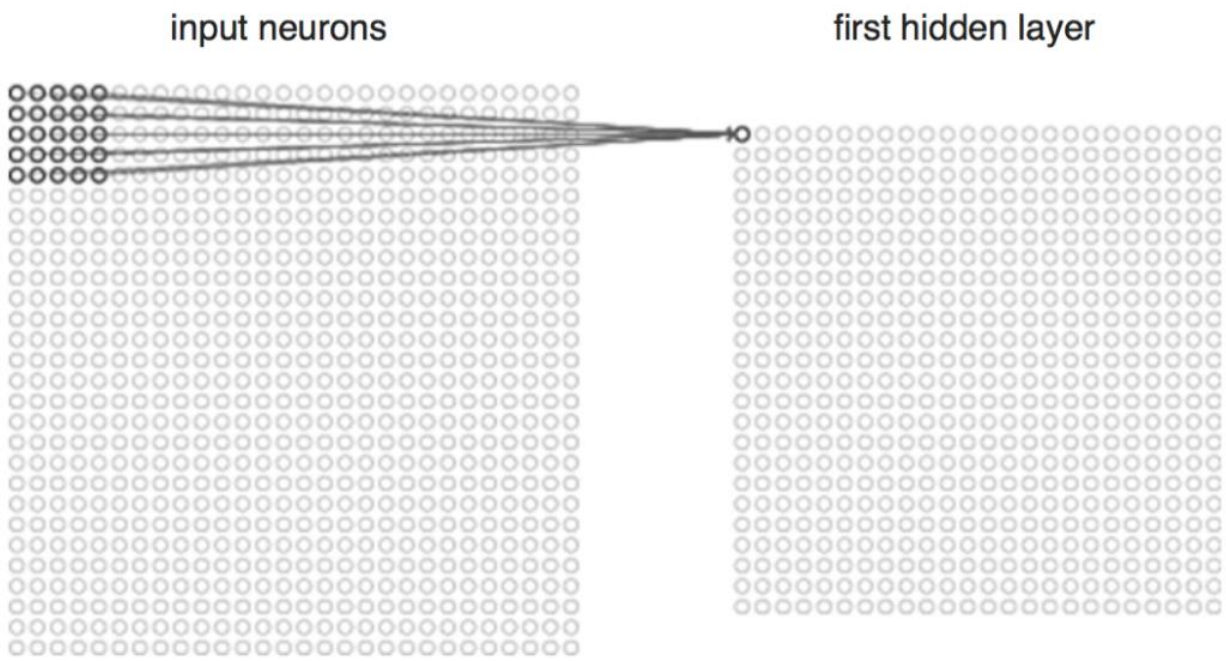

Figure 2. Convolutional operation [13]

All the feature maps extracted after the convolutional operation consists of many dimensions. To reduce the number of dimensions into a smaller matrix, a max-pooling layer is used. The pooling function selects the most important pixel by choosing the largest value of the pixels from a feature map. This layer helps in the learning and training process of the network.

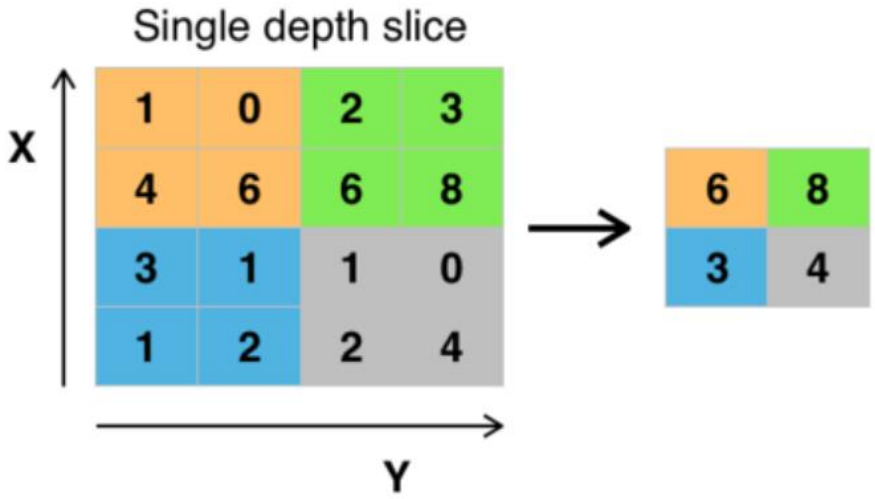

Figure 3. Max pooling operation [13]

Various CNN architectures have been developed over time to maximize the performance of image classification or feature extraction process. Some of the most famous models are AlexNet, VGGNet, ResNet, Inception/GoogleNet. In this project, pre-trained VGGNet consisting of 16 layers, also known as VGG16, is utilized to extract spatial features from the input. The VGG16 model is one of the very deep convolutional networks for image classification. It consists of a large number of hyper-parameters that take advantage of hierarchical information present in input data. Apart from classification, it is also used for semantic segmentation and action recognition.

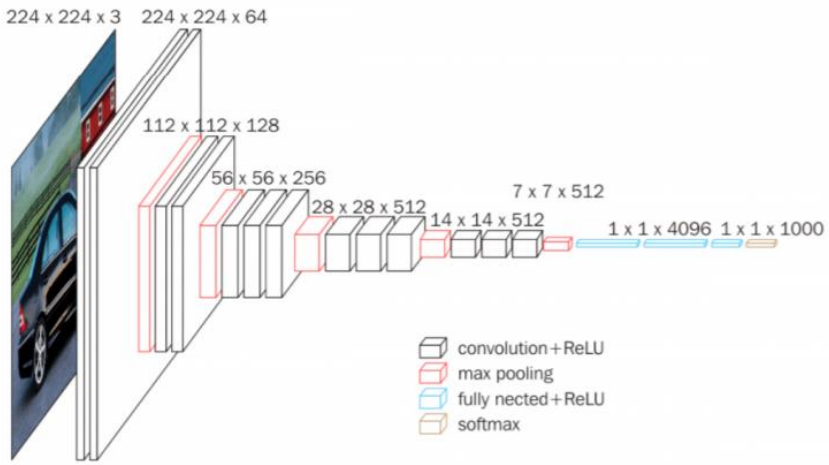

Figure 4. CNN – VGG16 [14]

It consists of 13 convolution layers (Conv) of 3x3 filters, same padding and with the stride of 1, followed by the ReLU activation function. It has five max-pooling layers (Pool) of 2x2 filters with the stride of 2. After the convolution and the max-pooling layers, the feature maps are passed into three fully connected layers (FC). These FC layers consist of most of the parameters which are used to classify or recognize any object. Since the model uses more than one ReLU activation function, it allows for better and more descriptive results. One of the disadvantages of VGG16 is the huge number of parameters (138 M) which requires more memory than other CNN models.

2.2 Recurrent Neural Networks – ConvLSTM

In sequential data or time-series data where the single input is dependent on the previous series sequence, a convolutional neural network (CNN) won't be enough. To extract information from this type of data, recurrent neural network (RNN) architecture is used. RNNs are used to extract temporal features in image classification, natural language processing, and speech recognition problems. The ability to process variable sequence and effect current input and output according to prior input, also known as internal memory, differentiates recurrent neural networks from other kinds of networks.

To overcome the exploding gradient problem of RNN, Long Short Term Memory (LSTM) was introduced in [15]. It minimizes the issue of exploding gradient problem of RNN. The LSTM models proved to be very effective in temporal problems, but it doesn't take spatial features into consideration which led to lack of spatial information. To overcome the problem of spatial redundancy in LSTM, ConvLSTM was introduced as an alternative [16]. Since this project utilizes the spatiotemporal features of the input data, ConvLSTM is used for affordance learning.

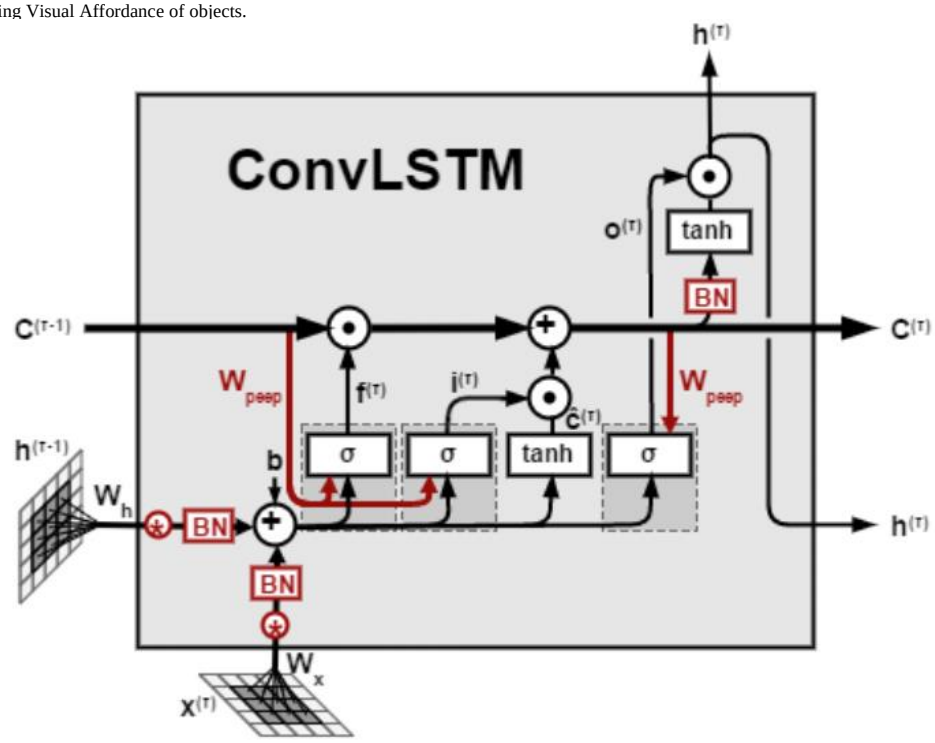

Figure 5. ConvLSTM cell [17]

To incorporate the spatial features of the input sequence along with the temporal features, all the input vectors X_t, cell states C_t, hidden states H_t and the gates i_t, f_t, o_t are 3D tensors who's last two dimensions i.e., rows and columns are spatial dimensions. It represents the input vectors and hidden states as vectors standing on a spatial grid. This change in architecture enables ConvLSTM to capture both spatial and temporal features. The following equations represent the internal calculations implemented in a ConvLSTM cell where '*' represents convolution operator and 'o' represents Hadamard product [16].

$$i_t = \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \circ C_{t-1} + b_f)$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tanh(W_{xc} * X_t + W_{hc} + H_{t-1} + b_c) \quad [16]$$

$$o_t = \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \circ C_t + b_o)$$

$$H_t = o_t \circ \tanh(C_t)$$

2.3 Fully convolutional Networks (FCN)

A fully convolutional network is utilized in the project to give predictions of human-object interaction regions as spatial heatmaps. The spatiotemporal learning from recurrent neural networks is transferred and applied to an image to give the respective output.

Convolutional neural networks (CNNs) are powerful deep neural networks to extract hierarchical features from the visual input data and solve the problem of image classification. To perform the same classification at the pixel level, fully convolutional networks were introduced [18]. The fully convolutional networks don't consist of fully connected layers which are used in CNNs to perform classification; instead, they rely on fully convolutional layers to classify each pixel of the image. In place of the Dense layer, FCNs consist of 1x1 convolutions, which perform the task of fully connected layers. Since FCNs efficiently learn dense predictions at pixel level classification, it is widely used in segmentation tasks [18].

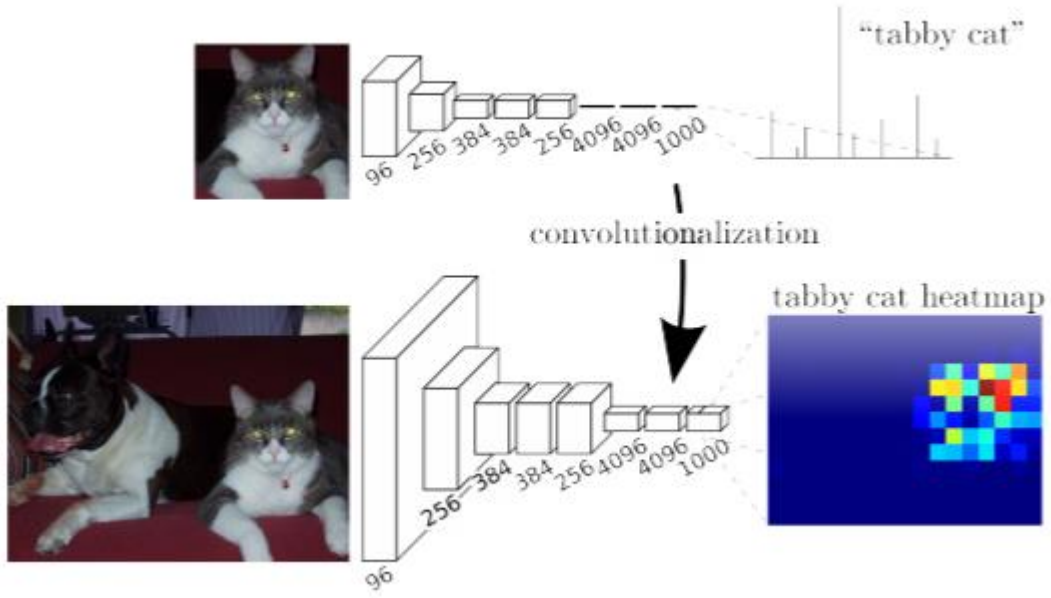

Figure 6. Fully convolutional networks [18]

FCNs take input of any size and combine the hierarchical features extracted from input data to produce dense course output maps of the same size. The output of each convolutional layer in

FCNs is a three-dimensional array consisting of h * w * d, where h and w represent the spatial dimension of the input size and d represent the feature dimension of the input image [18]. Since convolutional layers can recognize an object even if its appearance varies in some form, FCNs rely on spatial coordinates and local input regions.

2.4 Transposed Convolutional Layer

To upsample the feature maps to the respective input size, we utilize transposed convolutional layer in the project. Convolutional layers in a fully convolutional network reduce the size of the 3-D input tensor as the tensor passes through the max pooling and striding process. An upscaling operation is required to generate the same feature map of size greater than or equal to the input size. This is achieved by leveraging a transposed convolutional layer. Often Transposed convolutional layer is confused with the deconvolutional layer. The deconvolutional layer reverses the output of the convolutional layer and produces the original input visual data. The transposed convolutional layer manipulates the convolutional layer only dimension-wise. It goes in the opposite direction of the normal convolutional operation and produces an output of the feature map with a higher dimension.

Transposed convolutional layer performs the same stride and pooling operation as the convolutional layer by swapping the forward and backward passes of convolution. As a result, it produces a different spatial feature map of the visual input data. Following is the figure explaining the process of a transposed convolutional layer. Here 's' and 'p' represent the stride and padding of the layer.

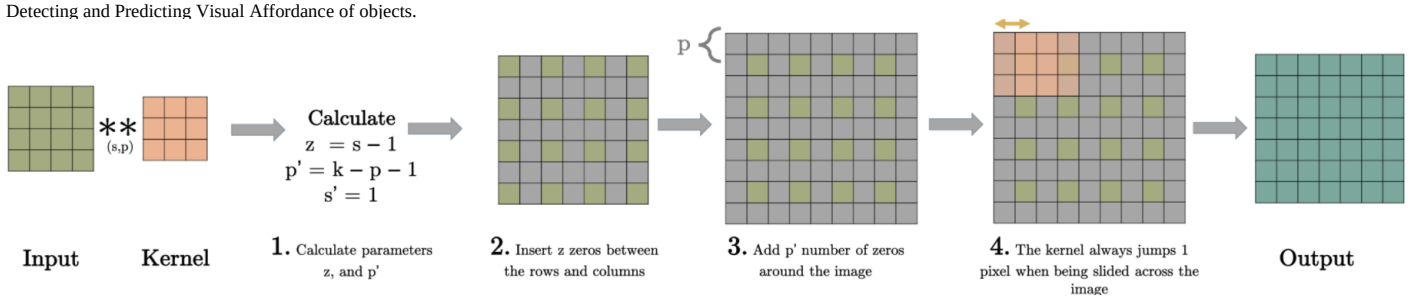

Figure 7. Transposed Convolutional Layer [19]

2.5 Attention Mechanism

The attention mechanism is leveraged in this project on top of ConvLSTMs, to better tune the output. In an encoder-decoder sequence to sequence model, the information of the whole sequence is encoded and put together into a single vector. The decoder takes this single vector of information as its input to give a translated output. This architecture works well for small sequences. As the length of the input sequence increases, the model is unable to retain the information of longer sequences. This results in information loss and low efficiency in the model. To overcome this problem, an attention mechanism is utilized.

Rather than considering all the weights of the vector, the attention mechanism focuses on the relevant part of the vector. It gives a different score to each weight depending on the context. These scores represent the relevance of a single weight with respect to the context. The rest of the weights are not discarded, they are not considered while producing the translated output. Through this mechanism, the decoder is able to capture global (or high level) information of the input sequence.

There are two types of attention mechanisms popularly used in deep neural networks, soft attention and hard attention. Soft attention is fully deterministic, whereas hard attention is stochastic. In soft attention, all the hidden states are taken into consideration and passed through the whole network. In hard attention, hidden states are sampled according to their score (or probability). The following picture represents the difference between soft and hard attention mechanisms.

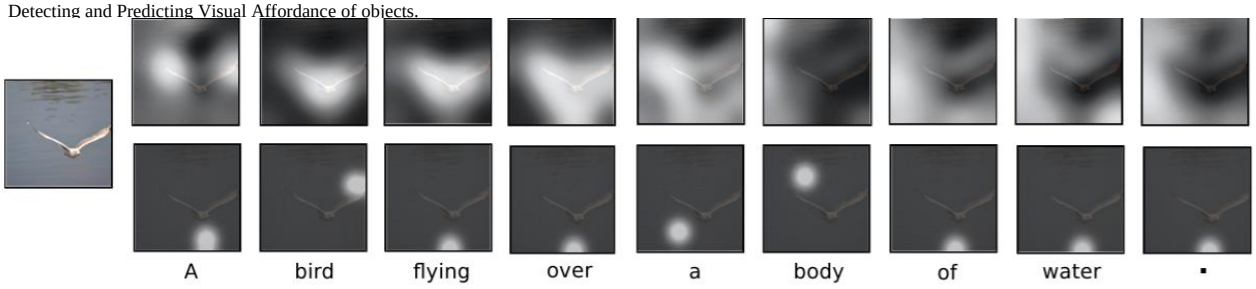

Figure 8. Attention over time [20]

(top row- soft attention, bottom row – hard attention)

2.6 Existing Approaches

Kuan, [11] developed a strongly supervised encoder-decoder architecture with ConvLSTM and spatial transformer networks as its encoder followed with deconvolution to find the affordance and interaction region between human and object. It had 40.79% classification accuracy on one of the popular datasets, OPRA (Online Product Review dataset for Affordance), and 2.34 Kl-divergence loss and 102.50 Negative Log-Likelihood loss for predicting interaction region. Since many existing approaches consist of heavy supervision, Tushar, [10] came up with a weakly supervised approach consisting of Convolution blocks and LSTM. This weakly-supervised approach performed better as compared to existing strongly supervised methods.

III. MODEL DESIGN

The main objective of this project is to learn the affordance of an object through frames of the videos. As a result, classify the affordance of a particular object and predict the human-object interaction region as a spatial heatmap. This chapter describes in-depth the deep learning neural networks and techniques used to develop the architecture of the model and how it helped in gauging the necessary features to perform action classification and heatmap prediction.

The model is based on an encoder-decoder architecture. The encoder takes the frames of the video as the input and produces an embedded vector for each video. The embedded vector is then used by the decoder to classify action and to predict heatmap. The decoder is composed of two models, one used to classify action label and the other to predict interaction region spatial heatmap. Both of the tasks are trained in a multi-task (jointly) manner.

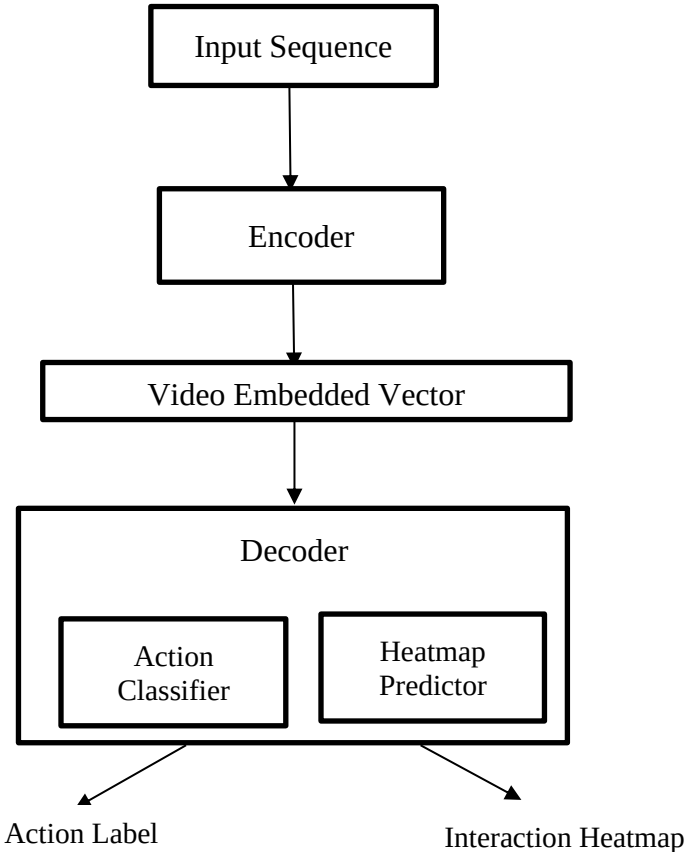

Figure 9. High Level Architecture

3.1 Encoder

In a video, the interaction between the user and the human happens for a short while. So, to extract the necessary visual cues from the video, convolutional LSTM networks (ConvLSTM) are used. In the encoder, we handle the motion and the content of the video independently, as it extracts the temporal dynamics and the spatial features of the video respectively, over time. The motion of the video is captured by recurrently taking element-wise subtraction of frames at time t and t-1. The content of the video is extracted from a single frame at time t.

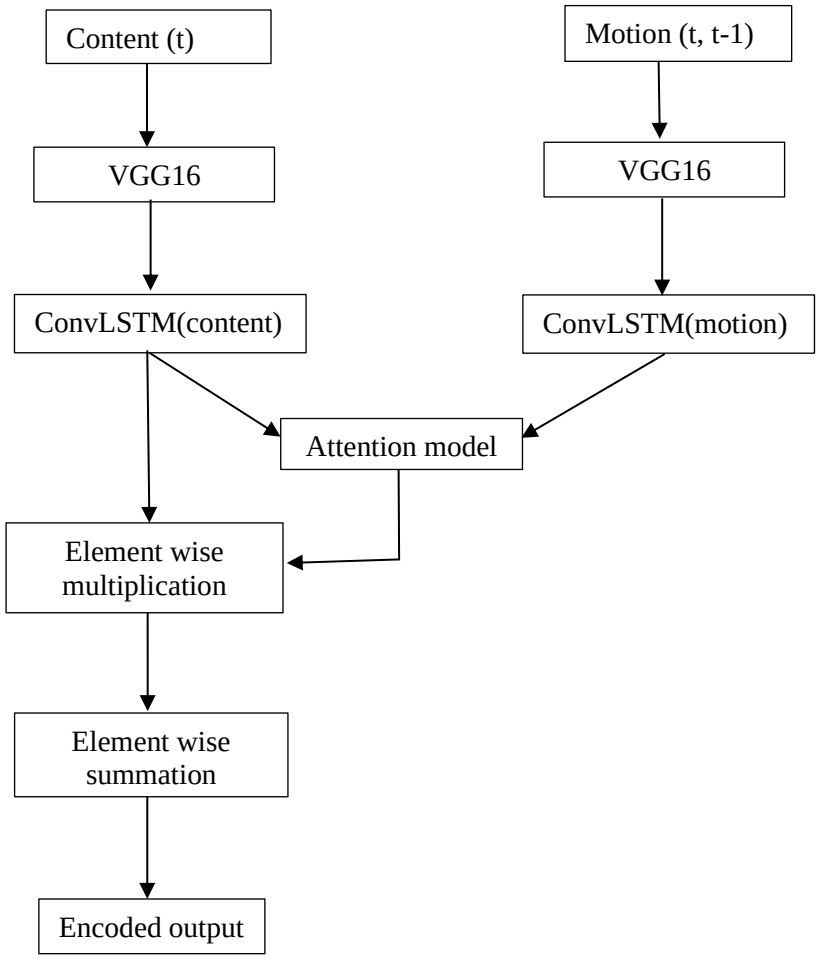

Figure 10. Encoder Architecture

The encoder takes two inputs: motion and content of the video. First, both of the inputs are passed through a pre-trained VGG16 (up to the last pooling layer), which acts as the visual feature extractor. These extracted features are then fed into each ConvLSTM network. The ConvLSTM extracts the spatio-temporal features of the video. To aggregate the output of the ConvLSTMs, an attention mechanism is used on top of the ConvLSTMs. This mechanism extracts the features which are relevant to the current context. The attention model is composed of a densely connected neural network layer which is followed by an activation layer to normalize the output.

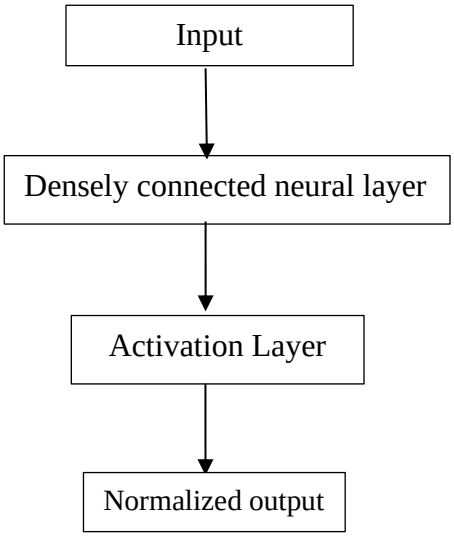

Figure 11. Attention model

The outputs from the ConvLSTMs are concatenated and passed into the attention model to compute an attention score at each time step t. The computed attention scores at each time step are multiplied with the output of ConvLSTM (content) in an element-wise multiplication manner. This helps us focus on the relevant features. Further, to obtain the final encoded output of the encoder, these dot-feature feature vectors for each time step are added using element-wise summation. The output of the encoder is a low dimensional embedded feature vector extracted from the video.

3.2 Decoder

The decoder takes the low dimensional embedded feature video vector as its input to predict the action label and spatial heatmap of the given visual input. The decoder consists of two models, an action classifier and a heatmap predictor.

Action Classifier

The action classifier takes the embedded feature vector and utilizes an LSTM network followed by a densely connected neural network layer to predict the action label i.e., the affordance of the object. Since the embedded vector is a low dimensional feature vector, it is repeated for each time step. For example, if the video consists of ten-time steps, the vector is repeated ten times across each time step. The modified embedded vector is fed into the LSTM whose output is passed into the densely connected neural layer to predict the action label.

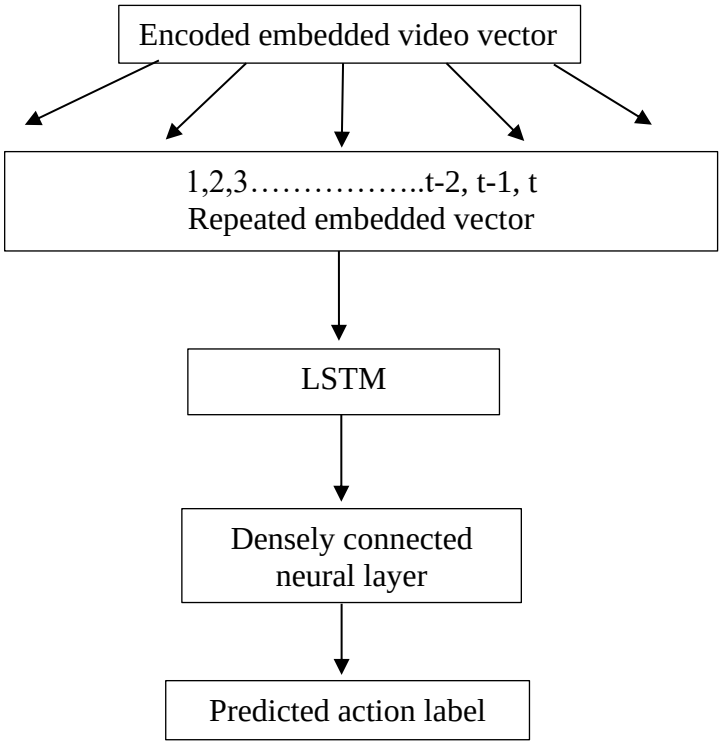

Figure 12. Action Classifier

Heatmap Predictor

The heatmap predictor predicts the spatial heatmap of the human-object interaction region. It takes two input, an embedded feature vector and an image on which the spatial heatmap output is predicted. The input image is first encoded by leveraging a pre-trained VGG16 (up to the last pooling layer). This encoded image is concatenated with the low dimensional feature vector. To get the predicted heatmap of the interaction region fully convolutional network (FCN) is used. The concatenated feature vector is passed through FCN. To obtain the predicted heatmap, the output of the FCN is passed into a transpose convolutional network to perform convolution in the backward direction, which upsamples the feature maps to the respective input size.

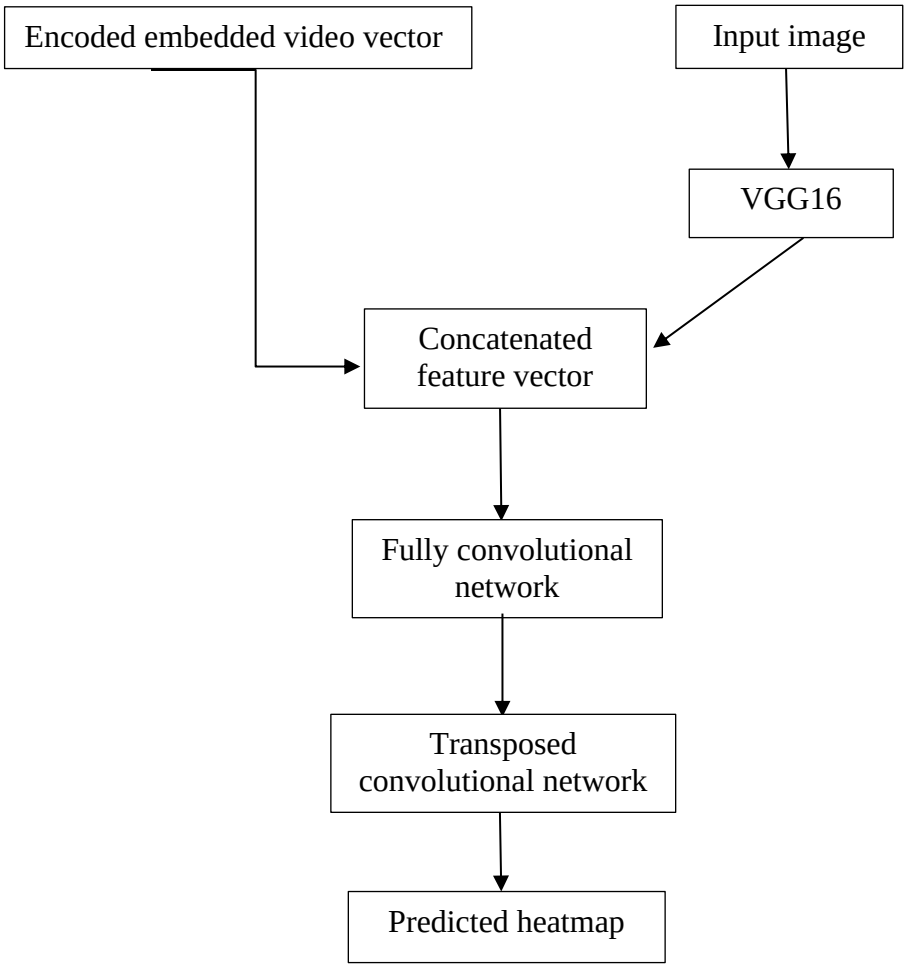

Figure 13. Heatmap Predictor

IV. IMPLEMENTATION

The base model consists of various sub-components. This chapter describes and focuses on the technical details of the computer vision techniques, data preprocessing, tools, the dataset used, environment details and; deep neural network layers used to implement the sub-components of the base model.

4.1 Dataset

The project's main objective is to implement a model that can learn affordances (i.e., actions) of an object from human-object interaction videos. The most common video datasets used for affordance learning are EPIC-KITCHENS [21] and Online Product Review dataset for Affordance (OPRA) [11]. EPIC-KITCHENS consists of first-person vision videos whereas, the OPRA dataset consists of third-person vision videos. So, for this project, we have used the OPRA dataset. The dataset is collected by scraping 11,505 demonstration clips and 2,512 object images from six popular YouTube product review channels. These channels include kitchenware objects, household appliances, consumer electronics, tools, and other objects [11]. It consists of seven action classes: Hold with 3992 videos, Put down with 849 videos, Touch with 9373 videos, Push with 2645 videos, Rotate with 1435 videos, Pull with 1138 videos, Pick-up with 1342 videos [11]. These scraped videos are further segmented into multiple small videos, where each of these small segmented videos consists of some human-object interaction.

Along with these segmented videos, up to five product (object) images are also collected to incorporate the affordance of the product. The affordance of a given task is incorporated by annotating 10 points on the product (object). It represents the interaction region and the type of affordance performed on that particular object. This is achieved by utilizing Amazon Mechanical Turk [11]. These annotation points are further utilized to compute a Gaussian blur ground truth

heatmap for the product image. A total of 20,774 video clips are collected, which are split into 16,976 videos for training and 3,798 for testing [11].

Figure 14.1 OPRA dataset (action label, video clip, annotated product image)[11]

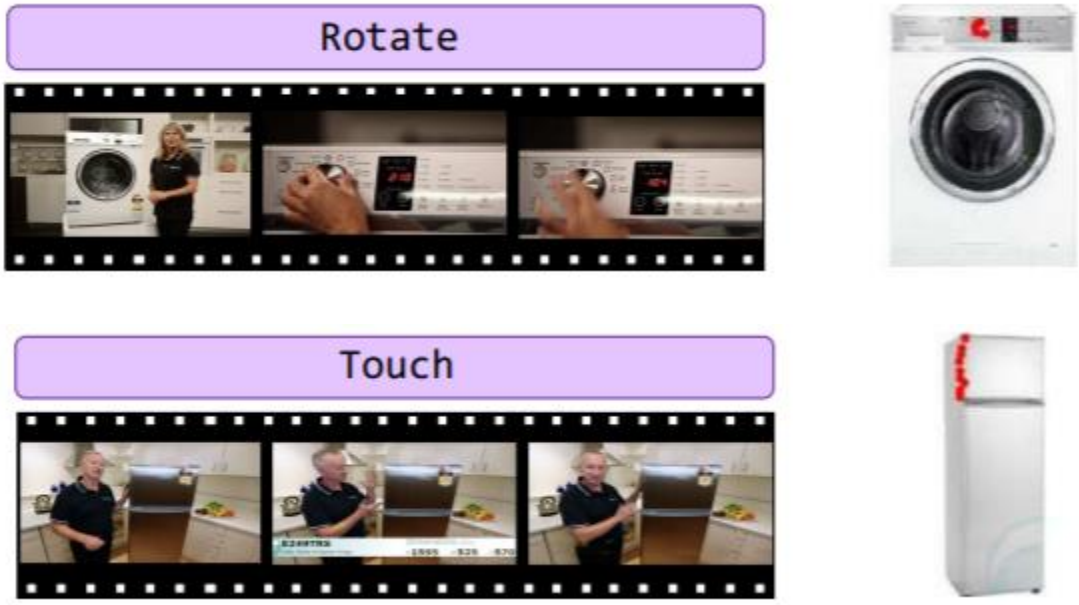

Figure 14.2 OPRA dataset (action label, video clip, annotated product image)[11]

4.2 Data pre-processing

OpenCV and FFmpeg are used to aid in data pre-processing steps. OpenCV is an open-source computer vision library, and FFmpeg is one of the popular software in handling video, audio, and multimedia data. Before feeding the data into the model, the following pre-processing steps were implemented.

1. From each demonstration video clip, 15 frames are generated at a one fps rate (frame per second) with the help of FFmpeg and FFprobe (a multimedia stream analyzer). All the videos with less than 15 frames are discarded. Since the model can't handle ragged input vector, the number of frames throughout all the videos are maintained uniform.

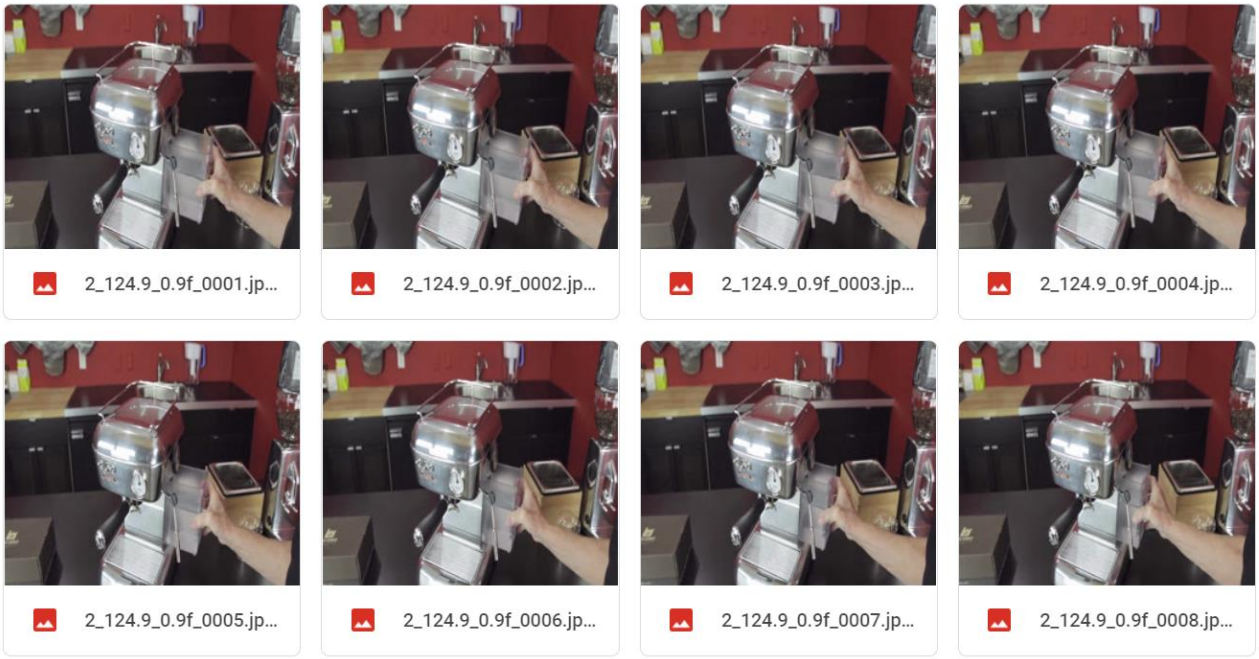

Figure 15. RGB sequence frames from video

2. The 15 RGB frames represent the content of the video over 15-time steps. To capture the motion of the video, a recurrent element-wise subtraction is taken between a frame at the t time step and a frame at the t-1 time step. A total of 14 such frames are generated. This is implemented with the help of the OpenCV absolute difference method.

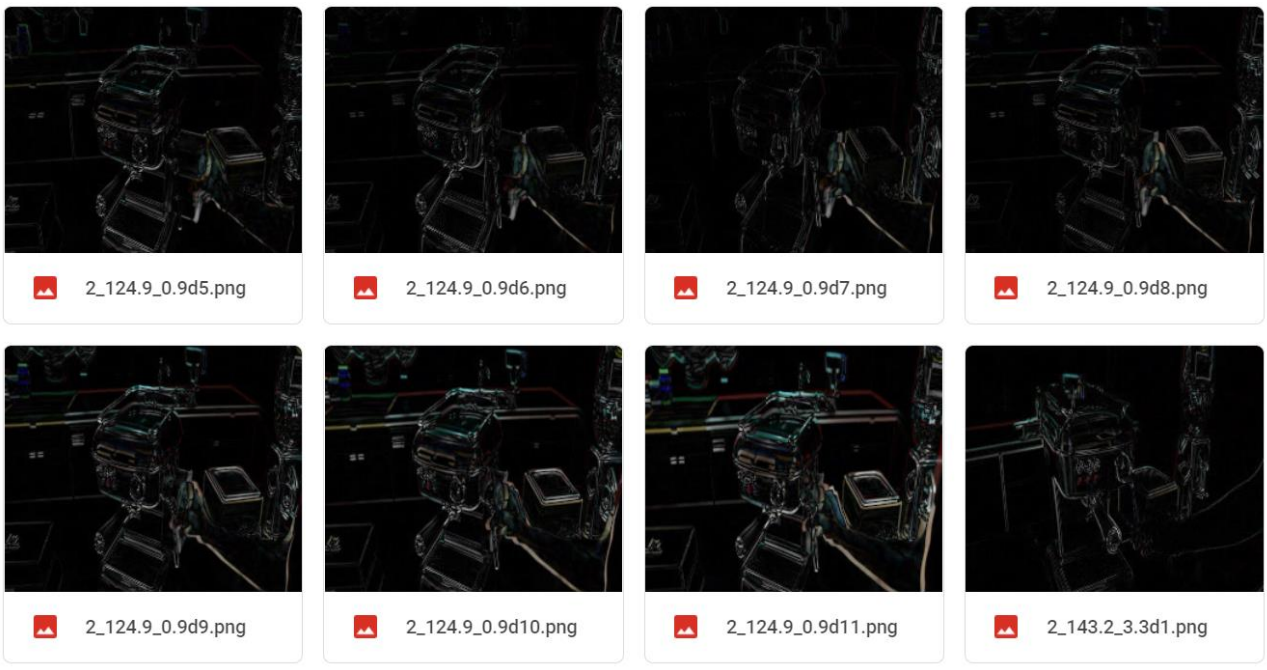

Figure 16. Motion frames from the video

3. All the RGB frames, motion frames and object images are resized to 128 by 128 dimensions.
4. The annotated points representing the affordance information are used to compute a Gaussian blur ground truth heatmap with the help of the Gaussian blur method present in OpenCV. The ground truth heatmap is saved in NumPy array of data type float 32 with dimension 128 by 128.

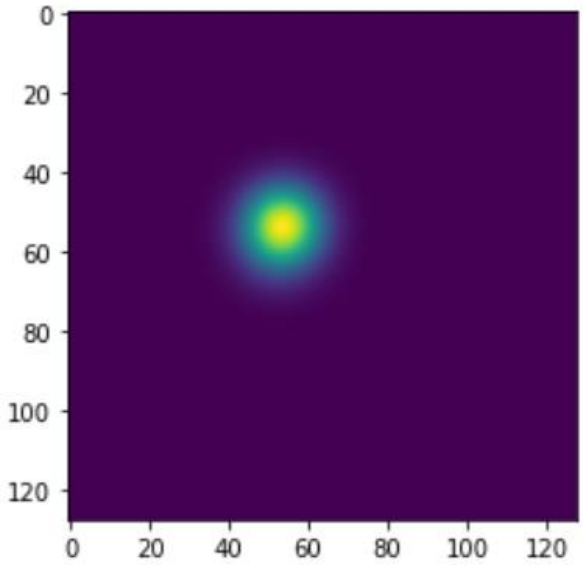

Figure 17. Ground Truth Heatmap (Gaussian blurred)

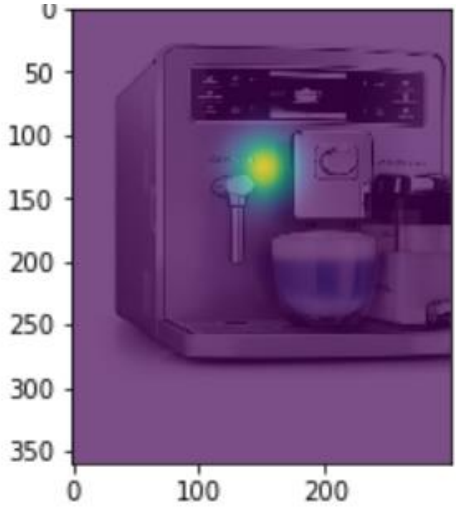

Figure 18. Ground Truth heatmap superimposed on product image

Since the base model takes three inputs (RGB frames, motion frames, and product image) and outputs two predictions (affordance label and heatmap prediction), a custom data generator is created to send the pre-processed data in the requisite format to the model. The custom data generator takes the pre-processed input, batch-wise, and yields a resized (128 by 128) visual data as output.

```
# yield the next training batch
yield [X_train_rgb,X_train_diff,X_images], [y_train,y_images]
```

Figure 19. Code snippet of custom data generator output

The X_train_rgb contains the RGB frames of the video, X_train_diff represents the motion frames of the video, X_images consist of the product image, y_train represents the affordance class the input belongs to and, the y_images consist of the ground truth NumPy array heatmap.

4.3 Environment Details

We have used Python programming language and Keras deep learning API of TensorFlow2 to build the whole model. To develop the model, we have used Keras's Functional API. The Functional API of Keras allows the creation of a model with multiple-input and outputs [22]. This allowed in manipulating the complex graph topology of the model. The project is running on Google Colab with a p3.2xlarge AWS EC2 instance as its backend. This EC2 instance consists of one NVIDIA Tesla V100 GPU along with 8 vCPUs and 61 GB RAM.

4.4 Model Implementation

The model consists of many sub-components: encoder, action classifier and heatmap predictor. The following architecture shows how these sub-components are interacting with each other.

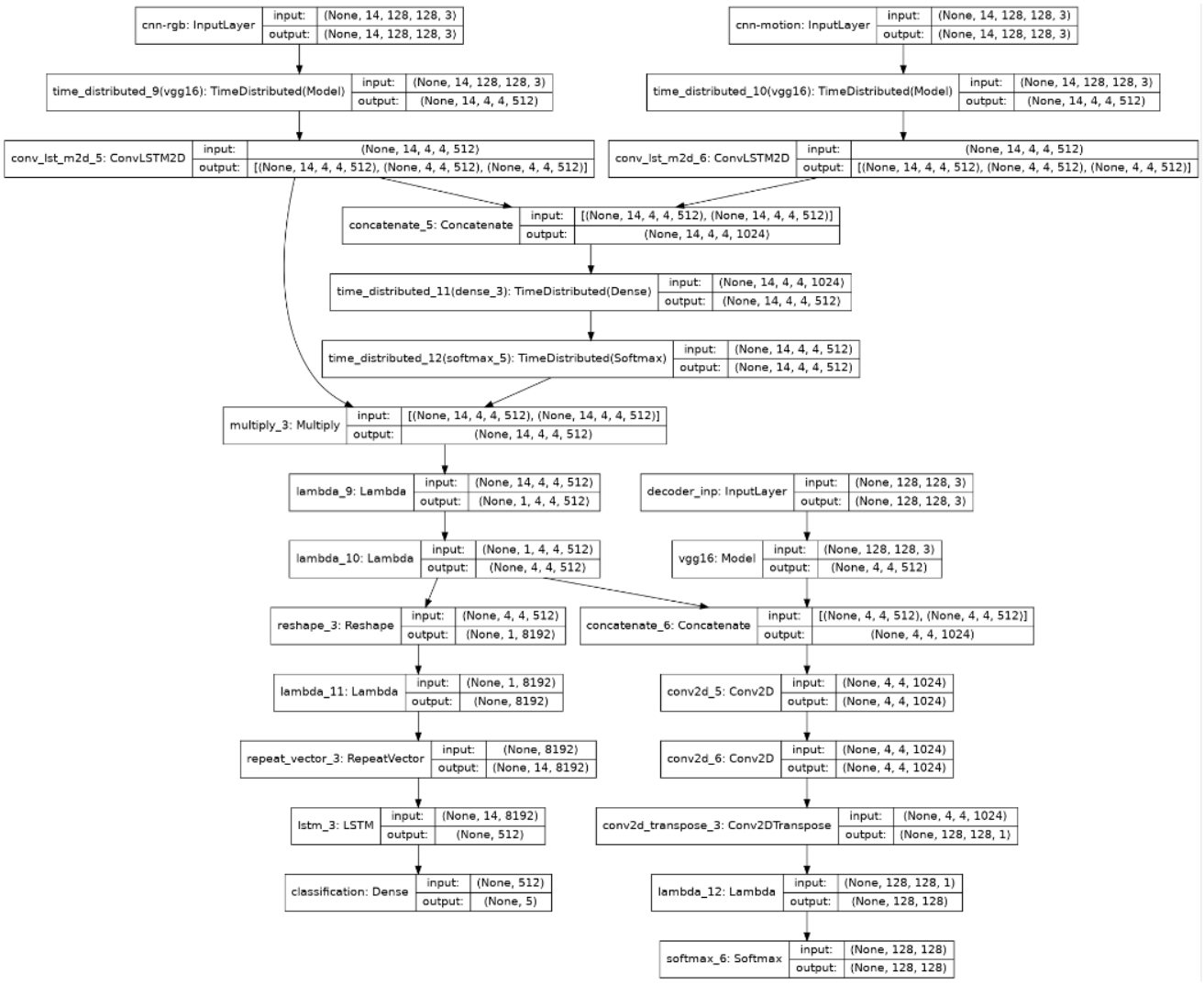

Figure 20. Model Architecture

The Figure 20. represents the architecture of the whole base model. It is implemented in an encoder-decoder structure. The model receives 14 RGB frames from timestep 2 till timestep 14.

Along with this model receives14 motion frames representing the difference of each consecutive frame. Each input is passed through a VGG16 pre-trained network with ‘imagenet’ as its weights. To encode and extract features from all the frames at each time step, a TimeDistributed layer is used. This layer allows to perform same computation on all the inputs individually. The two ConvLSTMs with kernel 3, strides 1 and ReLU activation, extract the spatial and temporal features from the sequence of RGB frames and the motion of frames respectively.

```
# RGB frames
Input_Spatial = Input(shape=(14,size,size,3), name='cnn-rgb')

cnn_spatial = TimeDistributed(vgg_model)(Input_Spatial)

encoder_1 = ConvLSTM2D(filters=512,kernel_size=3,strides=(1,1),padding='same',return_sequences=True, return_stat

encoder_outputs_1, encoder_state_h_1, encoder_state_c_1 = encoder_1(cnn_spatial)

# Motion frames
Input_Temporal = Input(shape=(14,size,size,3), name='cnn-motion')

cnn_temporal = TimeDistributed(vgg_model)(Input_Temporal)

encoder_2 = ConvLSTM2D(filters=512,kernel_size=3,strides=(1,1),padding='same',return_sequences=True, return_stat

encoder_outputs_2, encoder_state_h_2, encoder_state_c_2 = encoder_2(cnn_temporal)

concate_encoders = concatenate([encoder_outputs_1,encoder_outputs_2],axis=-1)
```

Figure 21. Encoder code snippet

After concatenating the output of both 512 feature channels in the last dimension, it is passed into the attention model. The attention model is composed of a dense layer followed by a softmax activation layer which normalizes the output and assigns the value in the range of [0,1]. The output of the attention model is the mask representing the score of each weight according to the context. To compute the mask at each time step, the TimeDistributed layer is leveraged. This mask is multiplied element-wise using a Multiply layer of Keras. The Multiply layer takes the input of the same size and outputs the vector of the same shape. With the help of Keras backend, the element-wise summation is taken along the time step axis which, produces a 4-D low dimensional feature vector as encoder’s output.

```
conv_attn = TimeDistributed(Dense(512))(concate_encoders)

soft_attn = TimeDistributed(Softmax())(conv_attn)
```

Figure 22. Attention model code snippet

```
multiplied = Multiply()([soft_attn,encoder_outputs_1])

Lambda_layer=Lambda(lambda x: K.sum(x, axis=1,keepdims=True) )

summation = Lambda_layer(multiplied)

sq_layer = Lambda(lambda x: tf.squeeze(x, axis=1))

summation = sq_layer(summation)
```

Figure 23. Encoder computation code snippet

The decoder consists of two models which perform two different tasks. One of them is the action classifier that classifies and predicts the affordance (action label) of the object. The encoder's output is flattened out and repeated over each time step using the RepeatVector layer. It repeats the input vector 'n' times. In our scenario, the flattened vector is repeated 14 times. The modified feature vector is fed into the LSTM with units 512 and dropout 0.2. This is followed by a dense layer which gives a list of probabilities as its output. The index with the highest probability is the affordance prediction of the input.

```
# Action Classifier

comb_features = Reshape((-1,summation.shape[1]*summation.shape[2]*summation.shape[3]))(summation)

sq_layer1 = Lambda(lambda x: tf.squeeze(x, axis=1))

comb_features = sq_layer1(comb_features)

lstm_inp = RepeatVector(14)(comb_features)

classifier = LSTM(512, dropout=0.2)(lstm_inp)

dense_layer = Dense(5,activation='softmax',name='classification')(classifier)
```

Figure 24. Action classifier code snippet

The heatmap predictor takes two inputs, product image and encoded embedded feature vector. The features from the product image are extracted by feeding it to a pre-trained VGG16 with 'imagenet' as its weights. These extracted image features are concatenated with the low dimensional embedded video vector. The concatenated vector is further fed into two consecutive fully convolutional neural layers with kernel size and strides as 1. The output of these fully convolutional layers is then passed into the Conv2dTranspose layer. This layer upsamples the output feature maps and performs the convolution in the other direction. The transposed output is passed through a softmax activation layer to normalize the output and assigns the value in the range of [0,1]. The final normalized (128 by 128) output is the predicted spatial heatmap for the given input.

```
# Heatmap Decoder

decoder_inputs = Input(shape=(size,size,3),name='decoder_inp')

decode_inp_image = vgg_model(decoder_inputs)

concatenate_features = concatenate([decode_inp_image,summation])

fcn1 = Conv2D(filters = 1024,strides=1,kernel_size=1,padding='same')(concatenate_features)

fcn2= Conv2D(filters = 1024, strides=1,kernel_size=1,padding='same')(fcn1)

upscaling = Conv2DTranspose(filters=1,kernel_size=64,strides=32,padding='same')(fcn2)

sq_layer2 = Lambda(lambda x: tf.squeeze(x, axis=-1))

upscaling = sq_layer2(upscaling)

outputs = Softmax()(upscaling)

model = Model([Input_Spatial,Input_Temporal,decoder_inputs],[dense_layer,outputs])

return model
```

Figure 25. Heatmap predictor code snippet

V. EXPERIMENTS AND RESULTS

To compile the model, we have used Adam optimizer with a learning rate of 0.0001. Adam optimizer is among the best adaptive optimizers and works well with sparse data[reference]. The loss function opted for affordance (action) prediction is categorical cross-entropy, and for human-object interaction, spatial heatmap prediction is KL divergence. The categorical cross-entropy computes the categorical loss among the true and predicted values whereas, KL divergence calculates the Kullback-Leibler divergence loss between the y-true and the y-predicted value. Along with this, the mean squared error (MSE) for the action prediction model is also calculated. The MSE computes the mean squared error between the predicted and the true value. Following experiments were implemented, with varying numbers of data samples and action classes. All the input frames and input images are resized to 128 by 128 dimensions.

5.1 Experiment 1

For this experiment, we chose three classes: Hold with 730 segmented videos, Touch with 800 segmented videos and Push with 775 segmented videos. As the dataset consists of uneven distribution of data samples among classes, we manually chose an almost equal number of data samples for each affordance class. The model is trained for 50 epochs with a batch size of 12.

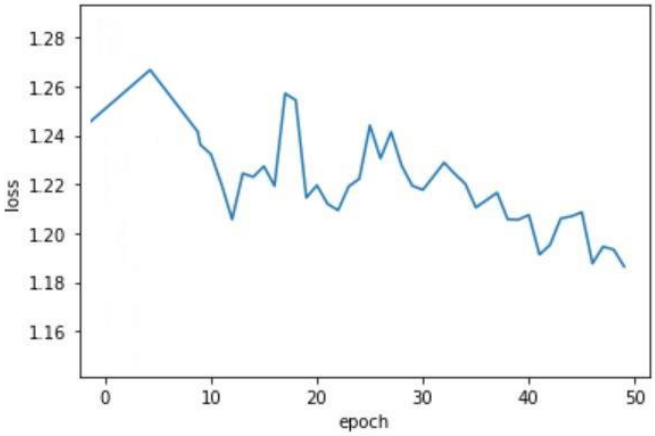

Figure 26. Overall training loss

Figure 26 represents the combined training loss of both the models: action classifier and heatmap predictor. The combined training loss is decreasing over time.

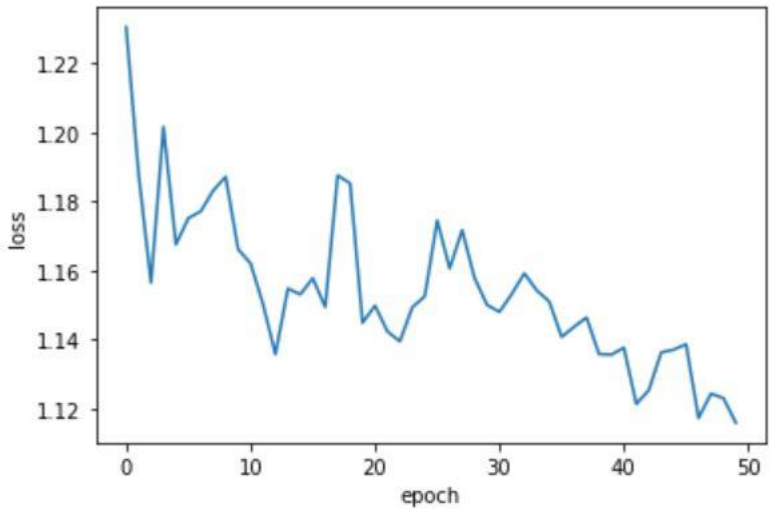

Figure 27. Classification training loss (Categorical)

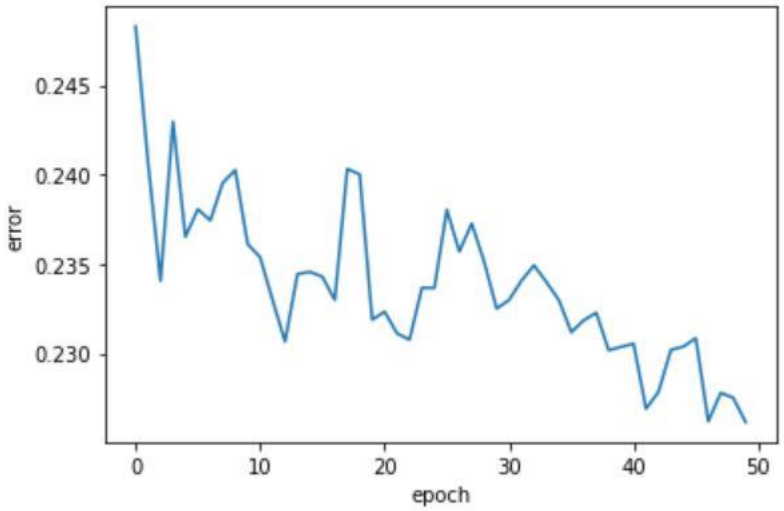

Figure 28. Classification training error (MSE)

Figure 27 and Figure 28 show the affordance (action) prediction loss and error over time. Both of them decrease in a similar manner over each epoch. On testing this model on unseen videos, it gave 35.4% classification accuracy and 6.97 KL divergence heatmap loss. Following are some of the predictions obtained from the model.

Ground Truth (Affordance Class) – Touch

```
print(prediction[0])
ind = np.argmax(prediction[0], axis=1)
print(action_labels[ind[0]])
```

```
[[0.27323252 0.38529786 0.3414696 ]]
touch
```

Figure 29. Predicted Affordance class

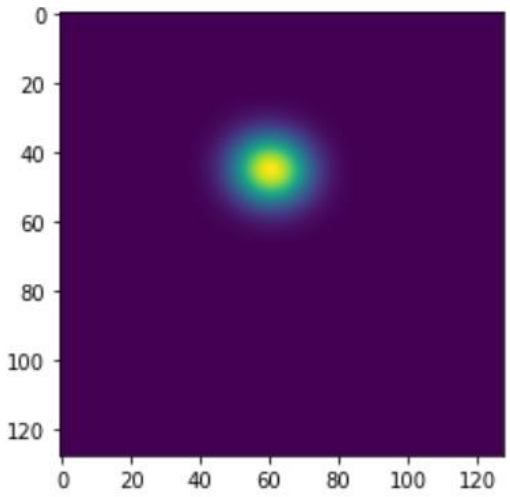

Figure 30. Ground Truth (Heatmap)

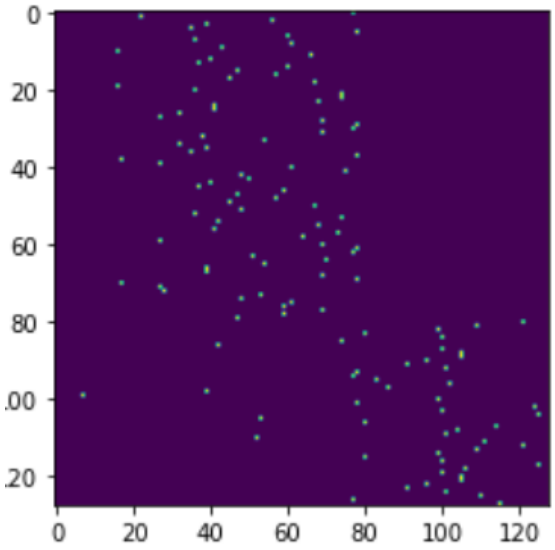

Figure 31. Predicted Heatmap

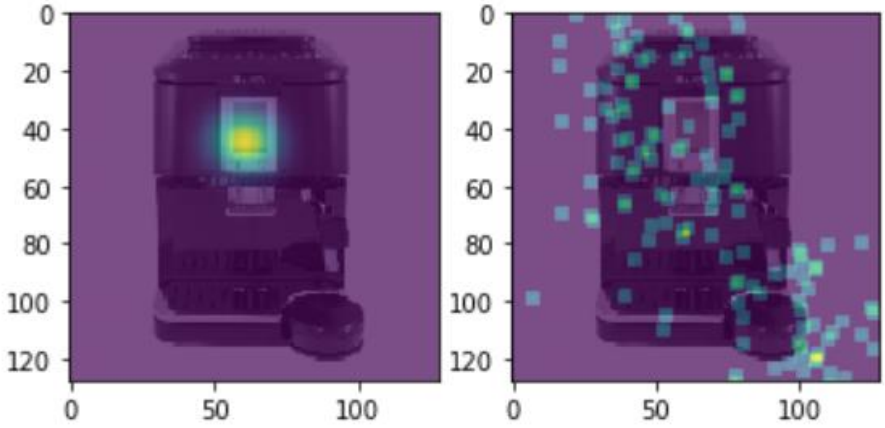

Figure 32. Superimposed actual and predicted heatmap on product image

For the affordance (action) class Touch, as observed from Figure 29, the action classifier is able to predict the class accurately. The predicted heatmap is composed of data points spread out throughout the image, but as we can see, the maximum number of the data points circulates the actual interaction region. So, the model was able to figure out and predict the actual region where the activity is happening. Figure 32 is achieved by applying Gaussian blur on the actual and predicted heatmap.

Ground Truth (Affordance Class) – Push

```
[100] print(prediction[0])
      ind = np.argmax(prediction[0], axis=1)
      print(action_labels[ind[0]])

      [[0.2202711  0.48487148 0.29485744]]
      touch
```

Figure 33. Predicted Affordance class

In this scenario, the action (affordance) classifier is not able to predict the affordance of the task correctly as the action push and touch are sort of similar in activity. So, the model is confusing the push action with the touch action. If we consider the top 2 actions with the highest probability, push comes among as second highest, as observed from Figure 33.

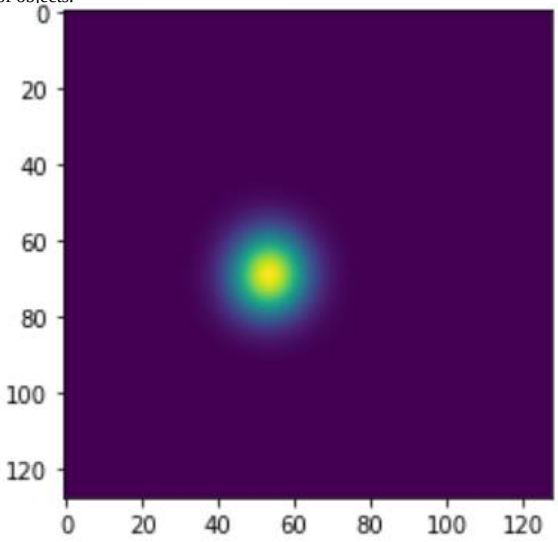

Figure 34. Ground Truth (Heatmap)

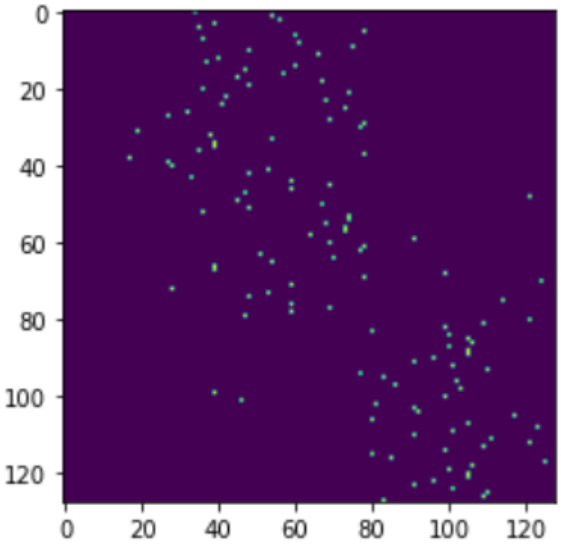

Figure 35. Predicted Heatmap

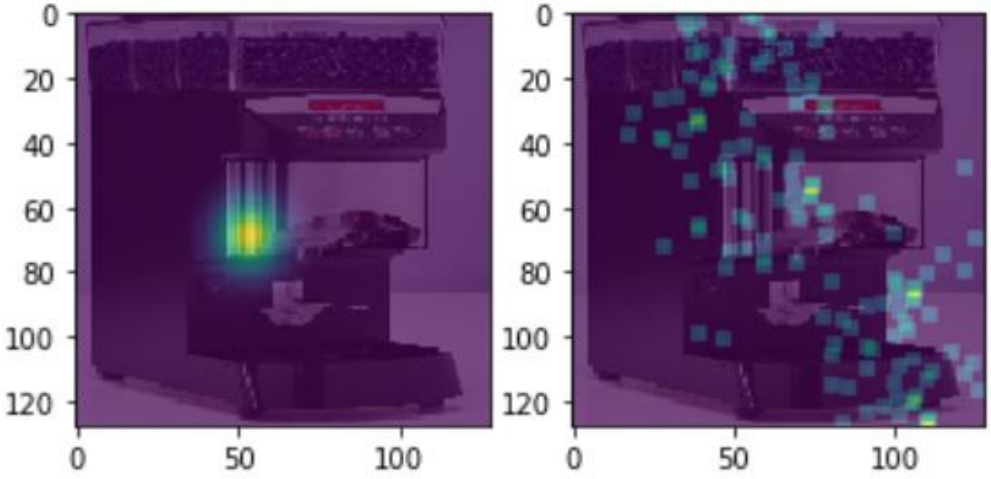

Figure 36. Superimposed actual and predicted heatmap on product image

Though the predicted affordance class is wrong, the heatmap is able to interpret the interaction region between the human and the object. As it can be observed from Figure 36, that the maximum number of data points with higher intensity are circulating the actual interaction region.

Ground Truth (Affordance Class) – Hold

```
print(prediction[0])
ind = np.argmax(prediction[0], axis=1)
print(action_labels[ind[0]])

[0.4385484, 0.31959203, 0.24185963]
hold
```

Figure 37. Predicted Affordance class

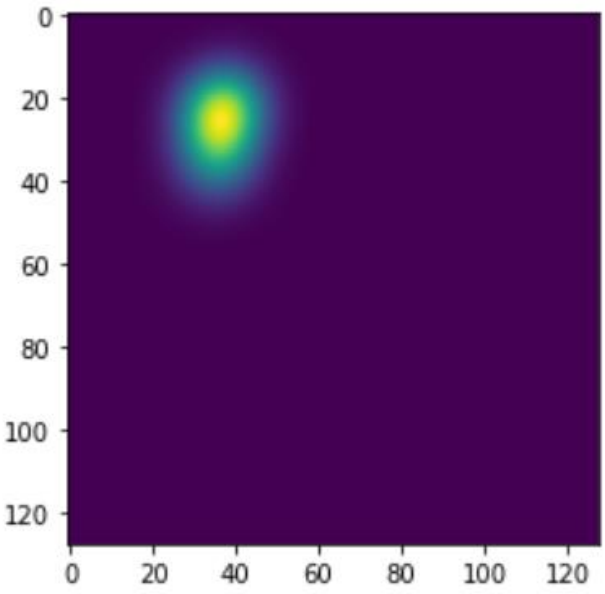

Figure 38. Ground Truth (Heatmap)

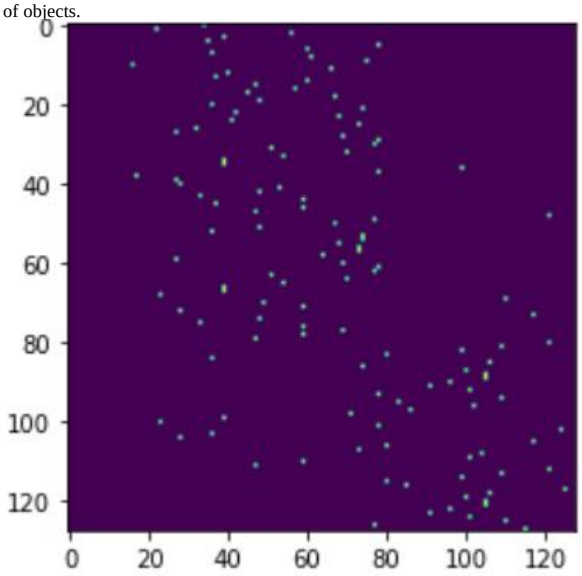

Figure 39. Predicted Heatmap

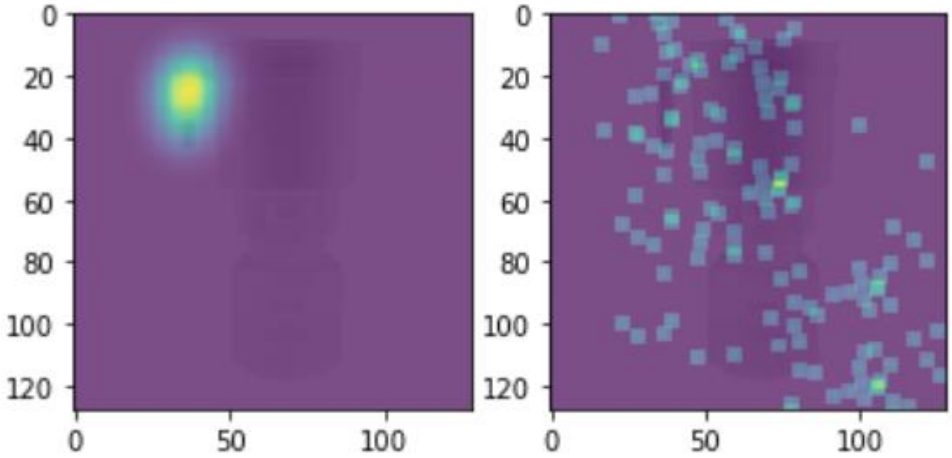

Figure 40. Superimposed actual and predicted heatmap on product image

The predicted heatmap consists of a lot of data points spread throughout the image. Rather than circulating the actual region, it's spread throughout the image. However, it is covering the region where actual activity is happening. The predicted affordance for this accurate, as seen in Figure 37.

Ground Truth (Affordance Class) – Push

```
[181] print(prediction[0])
      ind = np.argmax(prediction[0], axis=1)
      print(action_labels[ind[0]])

      [[0.31822214 0.32002962 0.36174828]]
      push
```

Figure 41. Predicted Affordance Class

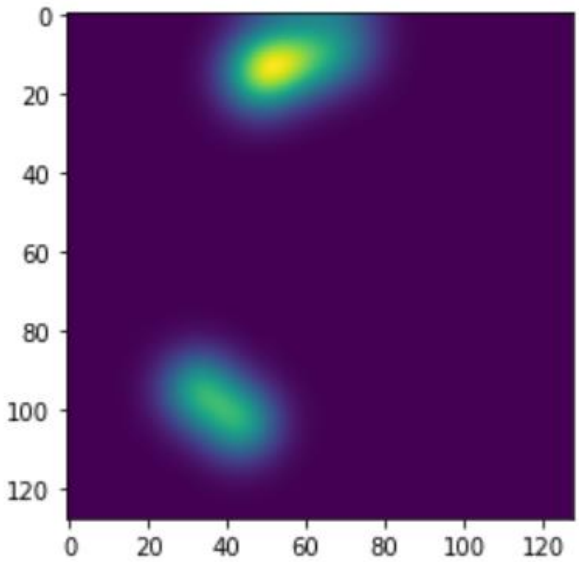

Figure 42. Ground Truth (Heatmap)

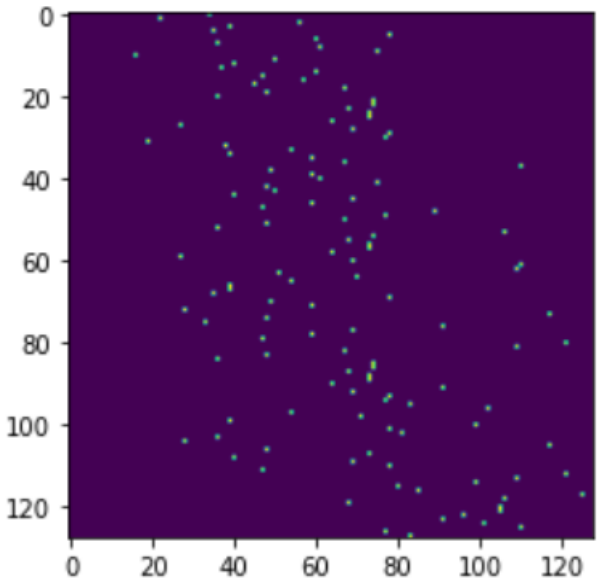

Figure 43. Predicted Heatmap

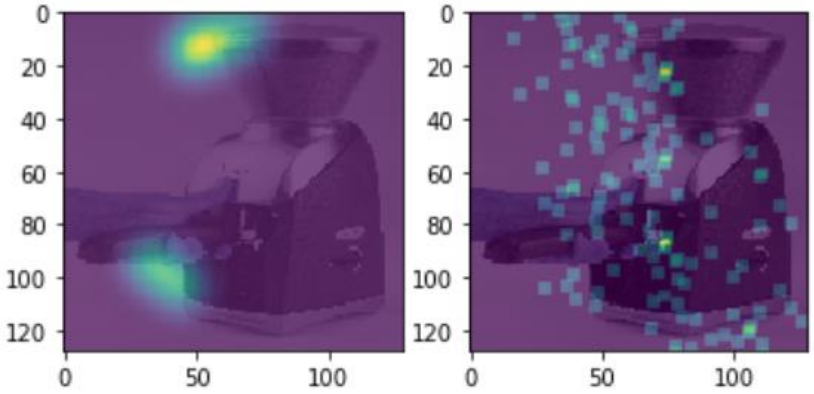

Figure 44. Superimposed actual and predicted heatmap on product image

This task contains two interaction regions, as observed from Figure 42. The predicted heatmap correctly predicted these two regions as all the data points circulate the two actual interaction regions where the activity occurs.

Ground truth (Affordance Class) – Touch

```
[194] print(prediction[0])
      ind = np.argmax(prediction[0], axis=1)
      print(action_labels[ind[0]])

      [[0.29831263 0.43870994 0.2629774 ]]
      touch
```

Figure 45. Predicted Affordance Class

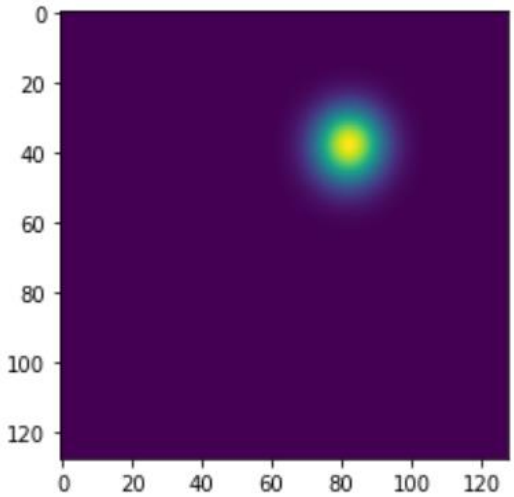

Figure 46. Ground Truth (Heatmap)

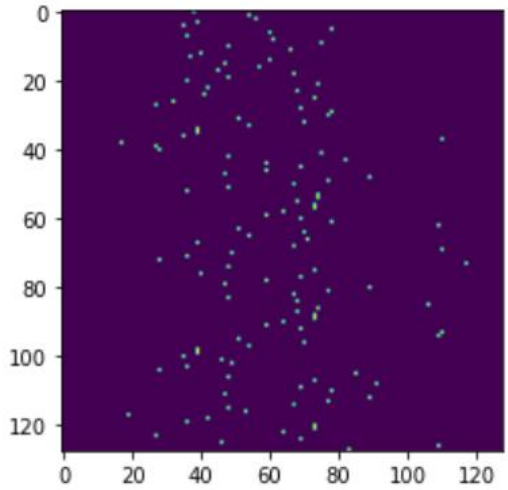

Figure 47. Predicted Heatmap

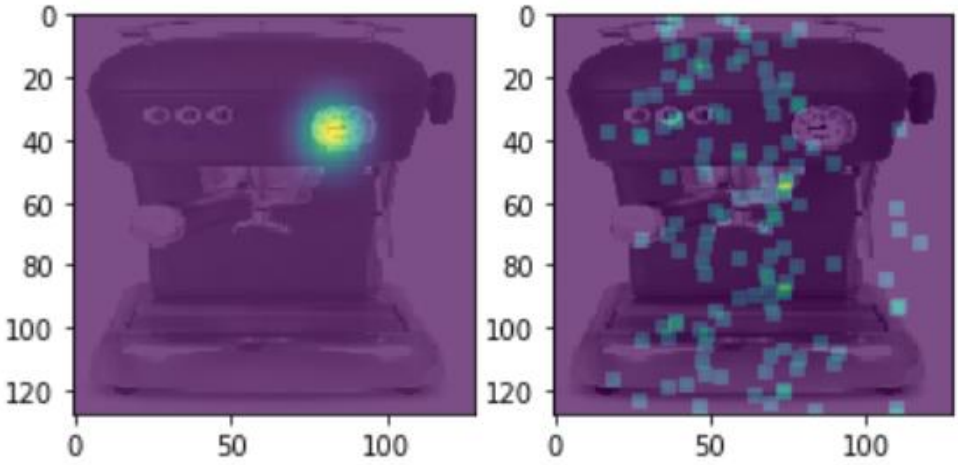

Figure 48. Superimposed actual and predicted heatmap on product image

As observed from Figure 45, the classifier accurately predicts the affordance class. However, the heatmap predictor didn't give accurate results. From Figure 48, we can see that the predicted data points are circulating different region than the actual interaction region.

5.2 Experiment 2

After observing the results from the previous experiment, we trained the model for a longer duration with a greater number of classes in this experiment. We run the model for 315 epochs with a batch size of 12 for five affordance classes: Hold with 415 segmented videos, Touch with 410 segmented videos, Rotate with 408 segmented videos, Push with 420 segmented videos, and Pick-up with 365 segmented videos. A total of 2018 segmented videos were used for training the model.

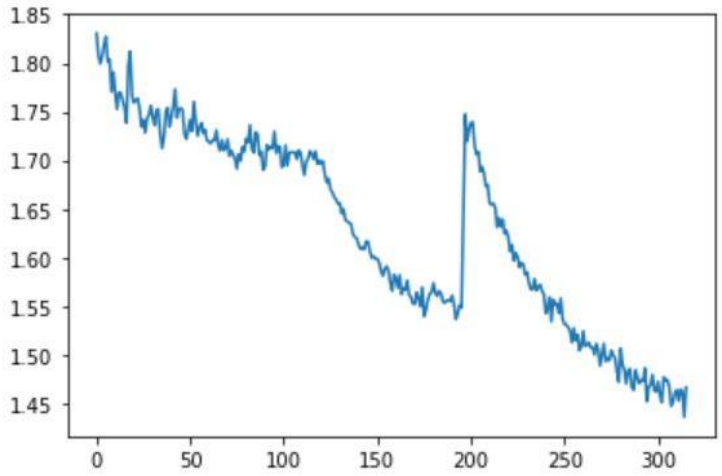

Figure 49. Overall training loss of the model
(x axis - loss value, y axis - no. of epochs)

As we can see in Figure 49, the loss increased between 200th - 201th epoch, though the overall training loss of the model decreased over time. The sudden increase and then decrease in the loss is due to a technical glitch that occurred in google colab while training. The connection between the AWS EC2 instance and the colab got disrupted for a while during training which led to this issue. As the model saves training after every epoch, due to the glitch, it could not save in time. This led to some loss in saving the training information of the model.

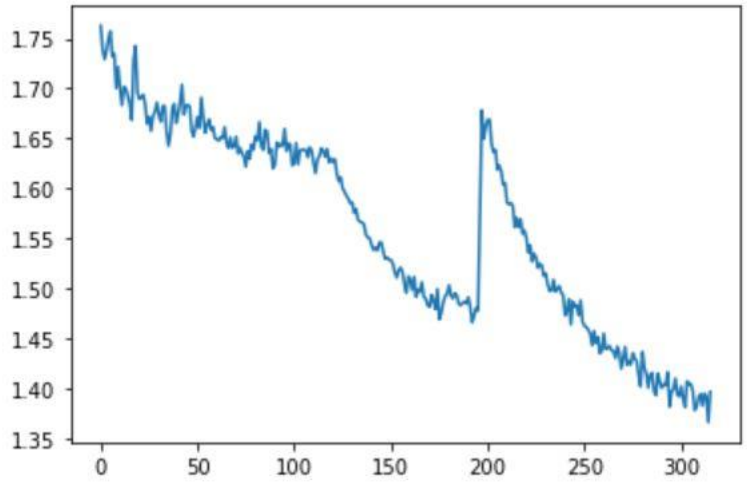

Figure 50. Classification training loss (Categorical)
(x axis – loss value, y axis – no. of epochs)

In Figure 50, the categorical training loss of the classification is decreasing over each epoch. It increases between 200^{th} - 201^{th} epoch, which is eventually reflected in the overall loss of the model. Similar behavior is noticed in computing mean squared error loss, as seen in Figure 51.

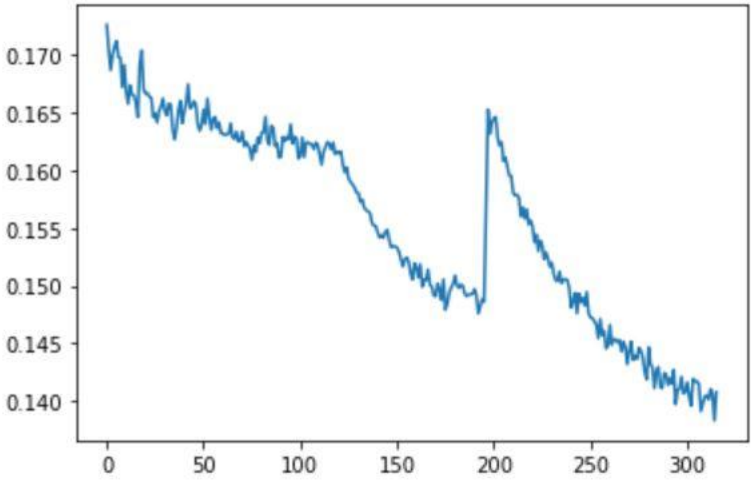

Figure 51. Classification training loss (MSE)
(x axis – error value, y axis – no. of epochs)

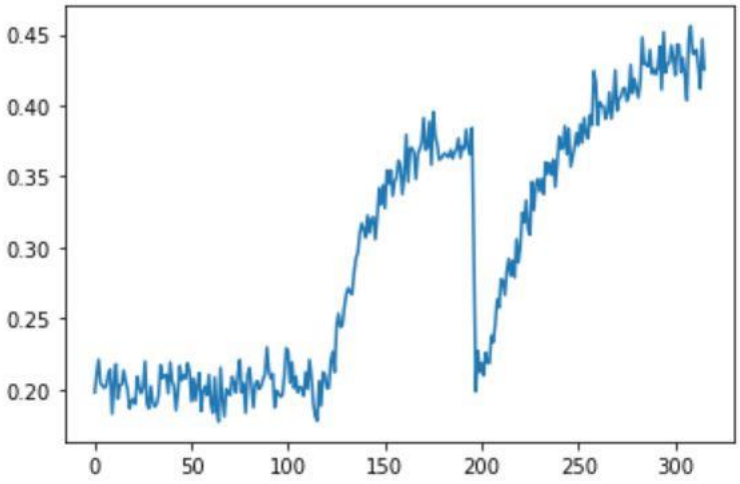

Figure 52 Classification training accuracy (Categorical)
(x axis – value, y axis – no. of epochs)

In the beginning, the accuracy of the classification is low till the 100th epoch. After that, the training accuracy increases till the 200th epoch, where the loss increased as observed earlier. After the loss fluctuation, the accuracy of the classification keeps on increasing over time.

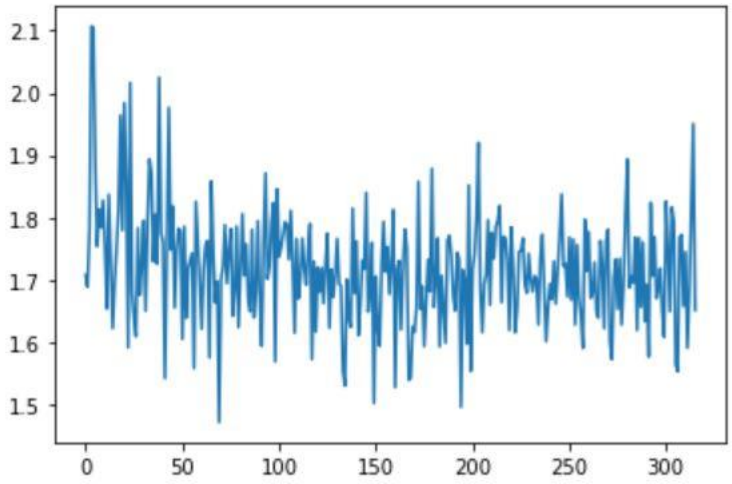

Figure 53. Overall validation loss
(x axis – value, y axis – no. of epochs)

The validation (testing) loss of the overall model decreases over time. It can be observed from the Figure 53, that the overall value of loss has decreased over each epoch.

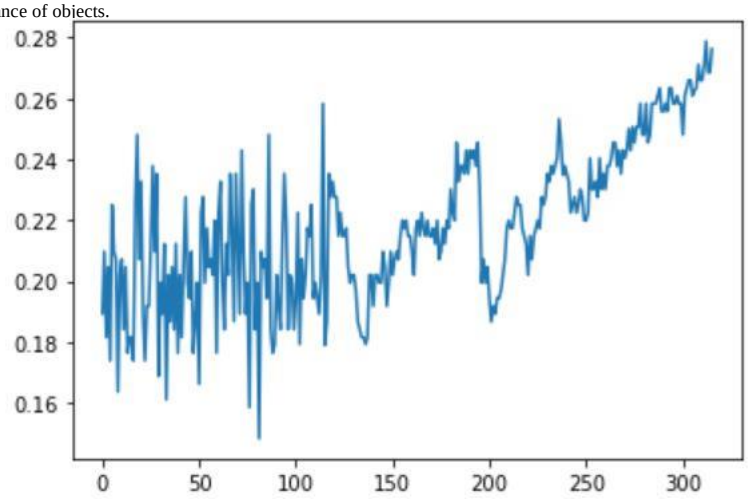

Figure 54. Classification validation accuracy
(x axis – value, y axis – no. of epochs)

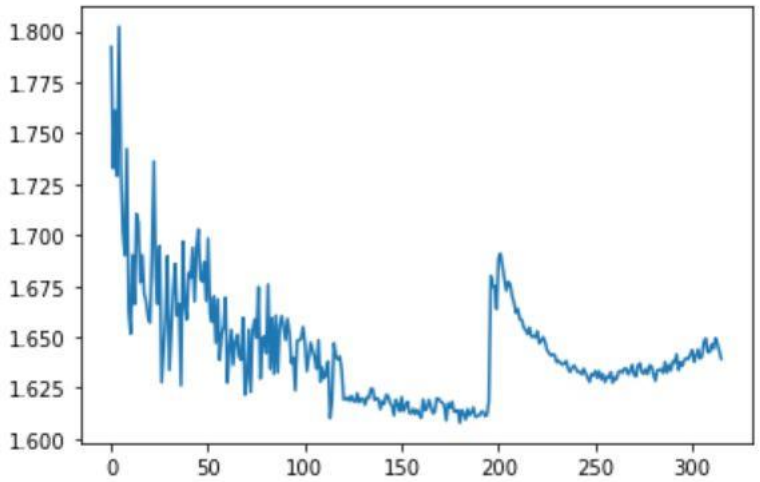

Figure 55. Classification validation loss
(x axis – value, y axis – no. of epochs)

The validation accuracy of the classification increases over time. Similarly, the validation loss of classification decreases over time. The sudden increase in the loss value at 200th epoch can be noticed due to the technical glitch discussed earlier. On testing the model for unseen data, it gave 48.48 % classification accuracy and 7.012 KL divergence loss for heatmap prediction. Following are some of the predictions and observations obtained from the model.

Ground truth (Affordance Class) – Touch

```
action_labels = {0: 'hold', 1: 'touch', 2: 'rotate', 3: 'pu
print(prediction[0])
ind = np.argmax(prediction[0], axis=1)
print(action_labels[ind[0]])
```

```
[[0.13899542 0.3731257  0.3403442  0.05909514 0.08843959]]
touch
```

Figure 56. Predicted Affordance class

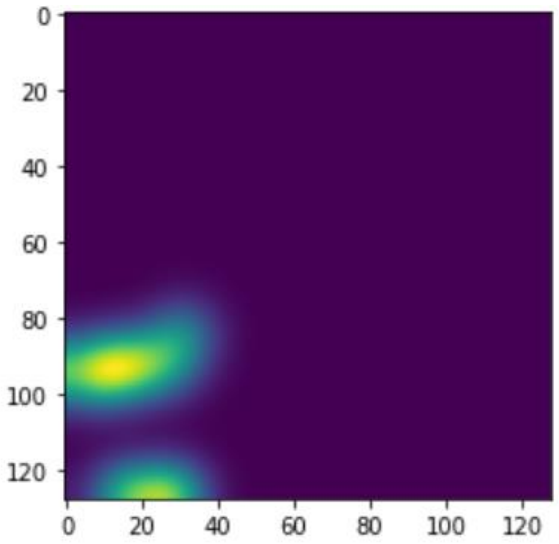

Figure 57. Ground Truth (Heatmap)

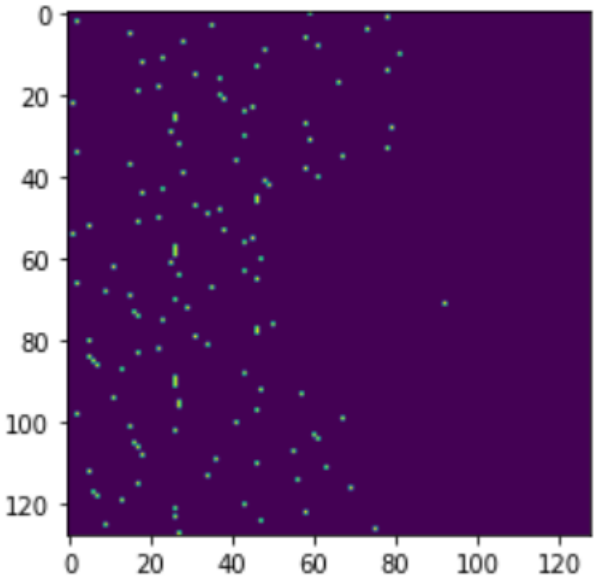

Figure 58. Predicted Heatmap

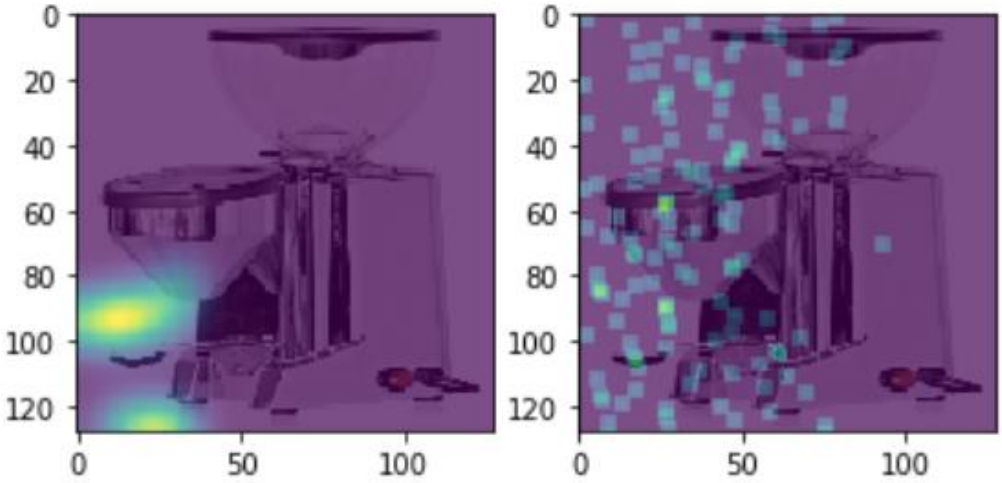

Figure 59. Superimposed actual and predicted heatmap on product image

For the above scenario, the classifier is able to predict the affordance class accurately. Also, the heatmap predictor is able to predict the actual interaction region as the maximum number of data points with the highest frequency are covering the actual interaction region.

Ground Truth (Affordance Class) – Pick up

```
[78] action_labels = {0: 'hold', 1: 'touch', 2: 'rotate', 3: 'push', 4:
     print(prediction[0])
     ind = np.argmax(prediction[0], axis=1)
     print(action_labels[ind[0]])

     [[0.10818663 0.17142057 0.16614896 0.20552713 0.34871662]]
     pick_up
```

Figure 60. Predicted Affordance class

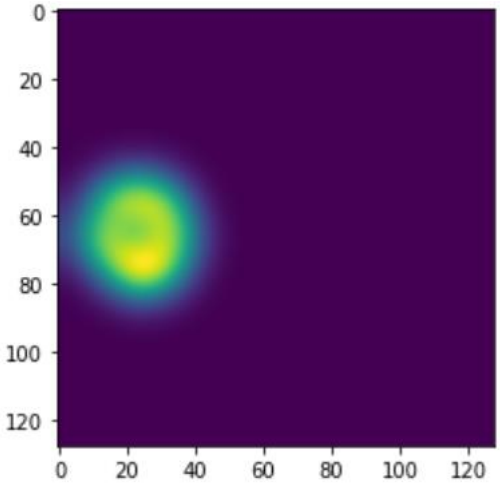

Figure 61. Ground Truth (Heatmap)

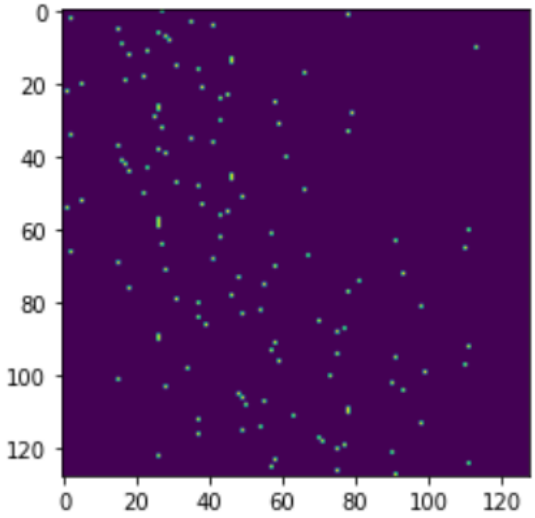

Figure 62. Predicted Heatmap

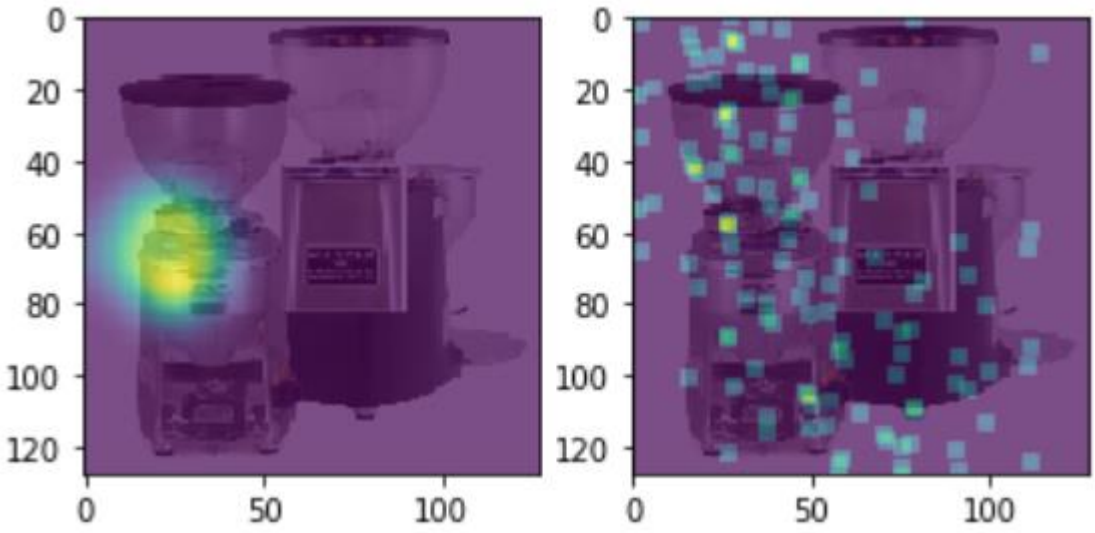

Figure 63. Superimposed actual and predicted heatmap on product image

From Figure 60, we can see that the classifier is able to predict the affordance class accurately. For the predicted heatmap, the heatmap predictor is able to predict the interaction region on one of the objects as present in the actual image. Also, the data points with high frequency are present on the actual interaction region and on one of the objects instead of both of them, giving us a correct prediction.

Ground truth (Affordance Class) – Rotate

```
[ ] action_labels = {0: 'hold', 1: 'touch', 2: 'rotate', 3: 'push',
    print(prediction[0])
    ind = np.argmax(prediction[0], axis=1)
    print(action_labels[ind[0]])

    [[0.08775322 0.19458954 0.4061065  0.25111073 0.06044002]]
    rotate
```

Figure 64. Predicted Affordance class

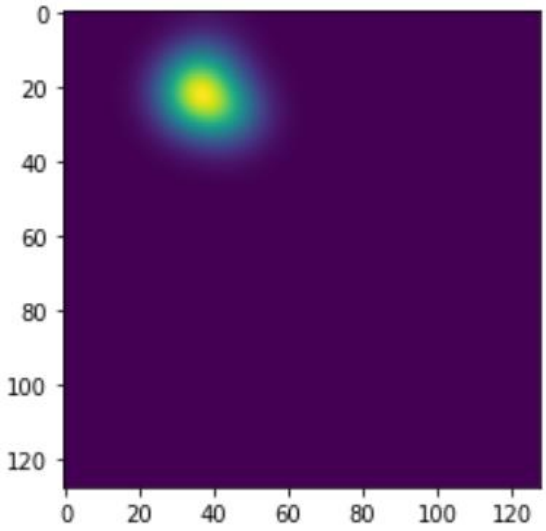

Figure 65. Ground Truth (Heatmap)

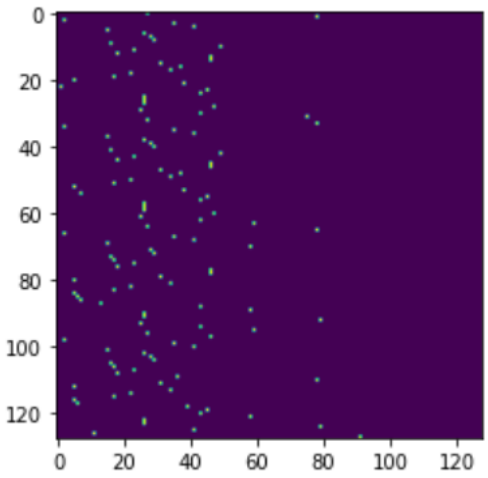

Figure 66. Predicted Heatmap

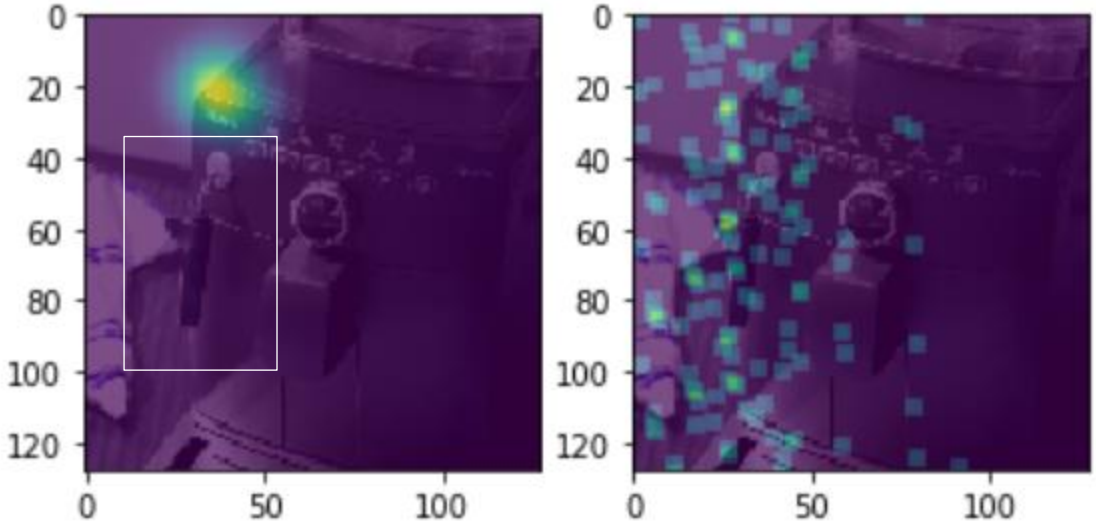

Figure 67. Superimposed actual and predicted heatmap on product image

The above input represents one of the issues present in the OPRA dataset. As we know that each product image has annotated points using which ground truth heatmap is generated. Some of the annotated points are plotted wrongly. The ground truth affordance class, rotate, for the above input is predicted accurately. As observed from Figure 67, the actual heatmap should be on the handle present in the white box; instead, it is on the top of the machine. In this scenario, our model is able to accurately predict the actual interaction region as the maximum number of data points are on top of the machine's handle.

VI. CONCLUSION AND FUTURE WORK

This report discusses how the model achieves affordance learning by taking advantage of spatial-temporal features from the visual data. To capture the spatial-temporal features of input video, CNN followed by ConvLSTM is used. An attention mechanism is utilized to enhance the extracted features, which tells what features are relevant and what to focus on. The model uses an action classifier and heatmap predictor to predict the affordance class and the human-object interaction heatmap of a given task. The heatmap predictor transfers the learnings extracted from ConvLSTMs to input product images and predict interaction spatial heatmap accordingly. From the experiments, we observed that the accuracy of the action (affordance) classifier increases with the increase in the duration of training. The heatmap predictor is able to predict better with more number data samples per class. The action classifier and heatmap predictor are trained together in a multi-task manner so that both the models can take advantage of the joint learning.

The results and observations obtained from the implementation give opportunities for possible future work. We can increase the accuracy of the heatmap predictor by collecting a higher number of data samples for each action class and training it on a lower learning rate. The action classifier needs to be trained for a longer duration to improve the accuracy. Better computed results can be achieved if machines with higher computation power and memory are used. Further, we can try changing the parameters within the model's layers along with different weights for pre-trained models to see if significant improvement is achieved.

REFERENCES

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

[2] A. Nguyen, D. Kanoulas, D. Caldwell and N. Tsagarakis, "Object-based affordances detection with Convolutional Neural Networks and dense Conditional Random Fields" in *International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 5908-5915.

[3] T. Do, A. Nguyen and I. Reid, "AffordanceNet: An End-to-End Deep Learning Approach for Object Affordance Detection" in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 5882-5889.

[4] A. Nguyen, D. Kanoulas, D. Caldwell and N. Tsagarakis, "Detecting object affordances with Convolutional Neural Networks" in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2765-2770.

[5] A. Roy, S. Todorovic, "A Multi-scale CNN for Affordance Segmentation in RGB Images" in *European Conference on Computer Vision*, 2016.

[6] J. Sawatzky, A. Srikantha, J. Gall, "Weakly Supervised Affordance Detection" in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2795-2804.

[7] B. Yao, J. Ma, and L. Fei-Fei, "Discovering object functionality" in *Proceedings of the IEEE International Conference on Computer Vision,* 2013, pp. 2512–2519.

[8] H. Koppula, R. Gupta and A. Saxena, "Learning Human Activities and Object Affordances from RGB-D Videos", in *The International Journal of Robotics Research*, 2013.

[9] T. Shu, X. Gao, M. Ryoo and S. Zhu, "Learning Social Affordance Grammar from Videos: Transferring Human Interactions to Human-Robot Interactions", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1669-1676.

[10] T. Nagarajan, C. Feichtenhofer and K. Grauman, "Grounded Human-Object Interaction Hotspots From Video", in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 8688-8697.

[11] 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)*, 2018, pp. 2139-2147

[12] M. Taylor, "Computer Vision with Convolutional Neural Networks", https://medium.com/swlh/computer-vision-with-convolutional-neural-networks-22f06360cac9#:~:text=Most%20computer%20vision%20algorithms%20use,and%20edges%2C%20from%20spatial%20data.

[13] Y. Upadhyay, “Computer Vision: A Study On Different CNN Architectures and their Applications”, https://medium.com/alumnaiacademy/introduction-to-computer-vision-4fc2a2ba9dc.

[14] R. Thakur, “Step by step VGG16 implementation in Keras for beginners”, https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c.

[15] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, in *Neural Computation*, 1997, pp. 1735–1780.

[16] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong and W. Woo, “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting”, in *Neural Information Processing Systems,* 2015.

[17] A. Xavier, “An introduction to ConvLSTM”, https://medium.com/neuronio/anintroduction-to-convlstm 55c9025563a7#:~:text=ConvLSTM%20theory&text=In%20such%20cases%2C%20an%20interesting,a%20Recurrent%20Neural%20Network%20architecture.&text=It%20is%20a%20Recurrent%20layer,are%20exchanged%20with%20convolution%20operations.

[18] J. Long, E. Shelhamer and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2015, pp. 3431-3440.

[19] A. Anwar, “Transposed Convolutional layer”, https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11.

[20] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel and Y. Bengio, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, in *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[21] EPIC-KITCHENS-100, https://epic-kitchens.github.io/2021.

[22] Keras, The Functional API, https://keras.io/guides/functional_api/.