Webcam Based Gesture Control Tool

Jeet Shah*
Email: jeet.s1@ahduni.edu.in

Kavyaa Sheth*
Email: kavyaa.s@ahduni.edu.in

Muskan Matwani*
Email: muskan.m1@ahduni.edu.in

*School of Engineering an Applied Science, Ahmedabad University

***Abstract*—The traditional way of presenting and interacting with our computers is cumbersome. We can leverage the use of current technology to make the user experience much better. In this document, we have proposed a way of interacting with computer using a real time recognition of dynamic hand gestures from video streams through webcam. The proposed solution uses 3D-CNN models for the live-video (gesture) classification purposes due to its superior ability of extracting spatio-temporal features within video frames. And based on this gesture detection, activities like scrolling, switching slides, zooming in and out, etc are automated using powerful python libraries. Additionally we have also used transfer learning with MobileNetV2 model to provide each user an option to customise their own gestures for each class of actions. We were able to achieve more than 90% accuracy in recognizing the custom gestures set by user in the real-time based system with our solution.**

***Index Terms*—Transfer Learning, Object Tracking, 3D CNNs, Virtual Mouse, Hand motion detection, YOLO, Classification, IaaS, SaaS.**

I. Introduction

The old method of interacting with computer using mouse and keyboard often times is a cumbersome task especially for a new user or a non technical person. The problem with this method is it is neither intuitive nor efficient. A very intuitive interface to interact with computers is to use hand gestures to perform tasks like changing slides, switching windows, zooming in and out, etc. This can be easily implemented using computer vision concepts like object recognition and tracking. Also the recent advancement in the field of AI like the concept of transfer learning can be exploited to customise the solution. Any AI algorithm training requires heavy computation, and that is where we can make use of cloud computing. The cloud computing architecture will provide a trained AI model to the end user.

II. Literature Survey

Existing literature suggests the application of Convolutional Neural Networks (CNN), which were originally used to classify images, in video action and activity recognition. In [2], a convolutional long short-term memory (LSTM) architecture is proposed in which features of video are first extracted by 2D CNN and later LSTM is applied for global temporal modeling.

State of the art CNNs such as Alexnet, VGG, GoogleNet, and ResNet had high computational costs and failed to capture long context with multi-frames, temporal information and motion patterns when applied for video classification task. To overcome this limitation, 3-D CNNs were proposed in [9], and have achieved the best performance for human action recognition from video on the recognition of human actions dataset.

There are several research works that addresses detection and classification using 3D-CNN models. The authors of [1] trained different 3D-CNN models on classic (huge) datasets to perform hand gesture detection. The authors of [7] proposed a different architecture that contains 2 parts - detector and classifier and trained the 3D-CNN model on the ego gesture and NVDIA dynamic hand gesture datasets. In [1], a comparison of different 3D CNNs is provided based on performance, FLOPs, speed(video clips/second) and parameters.

Model	MFLOPS	Params	Speed (cps)	Accuracy %
3D MobileNetV2	119	1.91M	243	91.96
ResNet-101	10612	83.29M	142	94.10
3D ShuffleNetV1	454	14.10M	88	92.56
3D SqueezeNet	728	2.15M	682	90.77

Table I
COMPARATIVE ANALYSIS OF DIFFERENT 3D-CNN MODELS IN TERMS OF FLOPS, NUMBER OF PARAMETERS, SPEED TO PROCESS VIDEOS PER SECOND AND ACCURACY

Of all the compared models, ResNet-101 gives the best performance in terms of accuracy but it requires a huge number of parameter and FLOPs, thus resulting in reduced speed. On the other hand 3D MobileNetV2 model requires lesser parameters to learn in comparison to other model with not much difference in accuracy. MobileNet is able to provide similar accuracy with significantly lesser parameters because it makes use of depthwise separable convolutions to construct lightweight deep neural networks(DNNs) [5]. The depthwise separable convolutions factorize the standard convolutions into a depthwise convolution followed by a 1x1 point-wise convolution. Compared to standard convolutions, depth-wise separable convolutions use between 8 to 9 times less parameters and computations. It also provides shortcut connections between the bottlenecks between the layers (Inverted Residual Block) which allows faster training and building deeper NNs.

The aforementioned approaches though achieving state-of-the-art performances, lack in giving user the flexibility of choosing their own gestures and instead fix the gestures according to the dataset the models they train on. Therefore we suggest the use of transfer learning on 3DMobileNetV2 to bridge this gap and give the user the flexibility to choose their own gestures to interact with the computer using predefined action classes.

In order to provide the user to control the mouse using hand gestures, we need to perform object tracking. The task of object tracking requires the algorithm to first detect the object. For this project we have explored two object classifiers - Haar cascade and YOLO. Haar Cascade classifier is a machine learning object detection algorithm based on the concept of features proposed by Paul Viola and Michael Jones in their paper "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.[10] YOLO is a real-time obect detection algorithm, which only processes the frame once and identifies all the object in it. Haar cascade is faster but pre-trained model of YOLO provides far better accuracy.

III. Implementation

A. Cloud Computing

*1) **REST API with Flask Framework** :* REST is an architectural style that defines a set or rules for a web application to send and receive data. Flask is a popular micro framework for building web applications. We built a REST API in Python using the Flask framework. The client (user) side script will first take data (gestures) from the user corresponding to the cursor control classes. The data is zipped and converted into base64 format to transfer its content in a json object. It then calls the service using HTTP post request by providing the endpoint URL. On the server side, a flask instance is created and an appropriate route for the service using the route() decorator to let our flask app know which URL should trigger that method. The method extracts the client data from the request body, decodes using base64 format and extracts zipped data content. Then, the model is trained on this data, and then zipped model returned in the response to the HTTP post request.

*2) **Apache Gunicorn SetUp**:* Gunicorn is a production WSGI server, i.e. it is the bridge between your Python application and the external world in production. The embedded WSGI server in Flask is not suitable to handle production scenarios. In addition to this, Apache2 is needed to fulfill the production level requirements of a HTTP server. To set up the Gunicorn server with flask, we instructed Gunicorn to start the application in the provided address providing the name of the wsgi file which will be the entry point. To set up gunicorn with Apache, we created a gunicorn configuration file which specifies the number of workers created and binds a unix socket file. Next, we created a systemd unit file which allows Ubuntu's init system to automatically start Gunicorn and serve Flask application whenever the server restarts. Then the service is started and enabled. To configure Apache to communicate the web requests to Gunicorn through the sock file, we first created Apache host configuration file which includes ProxyPass and ProxyPassreverse to pass the requests to the unix socket file we have created and configured with Gunicorn. In this file, we also change the limit of file size that can be passed through Apache to Gunicorn. Then, we enabled the new configuration and reloaded Apache.

*3) **Tunneling with AWS instance**:* We have used AWS to make our service publicly accessible. We acquired a remote server with Public IP by making a t2.micro(1GB Ram 1CPU) instance. The instance creation included generating private-public key pair to access the instance, generating security groups, enabling ssh access port forwarding, commands For http/s inbound traffic port forwarding and starting the instance. Then, we connected to an instance using PublicDNSName from above by setting appropriate permissions for the key. Furthermore, we securely routed the requests from our AWS EC2 Instance to our local machine using SSH Remote Port Forwarding. To achieve this, we created a tunnel from a port on the remote machine to a port on the local client machine. Moreover, since the public IP of the AWS instance is dynamic, we used dynamic No-IP configuration which assigned a hostname to the address of the AWS instance and helps in automatically updating the corresponding public IP address in case of changes. This made the service publically accessible which can be assessed by the hostname assigned by dynamic No-IP.

Figure 1 provides the overall Cloud Computing Management Architecture and the TABLE II elaborates the usage of the tools present in the architecture.

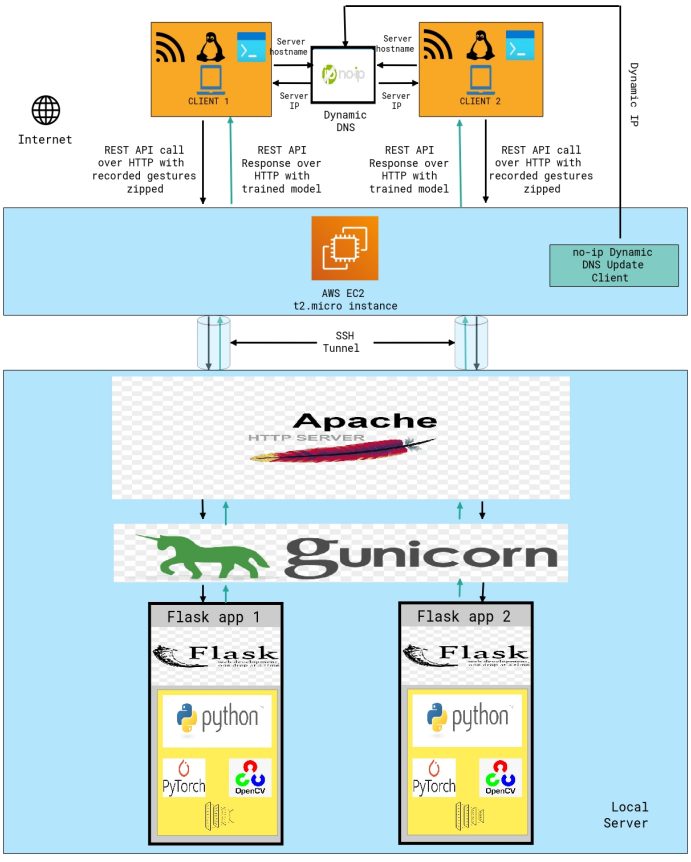

Figure 1. **Cloud Computing Management Architecture**

Tools	Functions
REST	HTTP based API to send and receive data from Server
No-IP DynDNS	Dynamic DNS Provider, it is used to avoid static IP charges
No-IP DUC	DNS update client that send latest IP Address to DNS
Apache2 HTTP server	Server Software to handle requests from multiple clients
Gunicorn	Local HTTP Production Server that maintains Flask app instances and distributes requests among them
Flask	Micro-framework to serve our Python code as an Web App
Python	Base Language for Code
Anti-virus	Antivirus to check received zip files for virus before extraction(Security)
OpenCV	Used for Hand tracking to get various statistics about movement
PyTorch	Framework to implement 3D CNN architecture
AWS EC2 Instance	Public facing server that is connected to local server via tunneling

Table II
TOOLS USED AND THEIR FUNCTIONS

B. Artificial Intelligence

*1) **Data Collection and Pre-processing on Client Side**:* The python script written on the client side first fetches the training data from the user that will be used to train the last layers of the pre-trained MobileNetV2 model. The python script allows users to do 5 different gestures corresponding to 5 cursor control classes for 10 times each. Additionally, another 10 frames are captured from the user where the user does nothing. This allows the model to be trained for the frames where the user is not making any of the 5 gestures and hence, this inclusion of the 6th class (do nothing class) helps to reduce the false positives by a greater extent. Furthermore, This will generate 60 samples of 60 frames each, captured at the speed of 30 frames/sec. Each sample is then downsampled from 60 frames to 16 frames in accordance to the model and stored with proper naming convention. After applying temporal transformation, the data is zipped and sent to the server in the request body of the REST API sent by HTTP request.

*2) **Data Augmentation, Hyper-parameter Tuning and Model training on the server side**:* The data sent from the client side is extracted according to the class name, sample number and the frame numbers. This data is further augmented to increase the dataset size to avoid under-fitting. The data augmentation is applied on the server side to reduce the amount of data sent by the client. Eight techniques are applied for data augmentation which include - blurring, centre crop, top left crop, top right crop, bottom left crop, and bottom right crop, changing contrast and changing brightness. After this, spatial transforms are applied on each frame to resize it to 112X112 in accordance with the model.

For transfer learning, we changed the last classifier layer of the pre-trained MobileNetV2 model to tune it according to our number of predefined user gesture classes. MobileNetV2 is trained on the 20BN-JESTER dataset. It is a large collection of densely-labeled video clips that show humans performing pre-defined hand gestures like thumb up, thumb down, sliding hand left, sliding hand right, shaking hand, etc in front of a laptop camera or webcam. This dataset contains over 1,50,000 videos having 27 classes of hand gestures[4]. We also tweaked the model implementing a dropout layer just before the updated classifier layer to prevent over-fitting.

To find the optimal hyper-parameter value for hyper-parameters learning rate and batch size, we took a brute force approach and plotted training loss, training accuracy, validation loss and validation accuracy for 30 epochs. These graphs help in determining the optimal value such that we could avoid over fitting of the model and save some disk space. The optimal value of 0.01 for learning rate and 8 for batch size was determined by this method.

For model training, we have implement Automatic Mixed Precision[11] method to reduce GPU footprint. Generally the training happens with float 32 operations, but float 16 operations can reduce memory usage with negligible accuracy drop. Auto-casting was used to achieve this. Also to prevent underflow with float 16 operations, gradient scaling was used in backward pass. We reduced the memory footprint by 30%.

The model is then trained with the augmented dataset and tuned hyper-parameters and then sent to the client side to further perform the live gesture detection.

*3) **Inference Part on the Client side**:* The application supports the following functions - horizontal and vertical scrolling, zooming in and out, selecting text, selecting file, copy, paste, changing slides, exiting full screen mode, and switching windows. To perform these actions based on gesture recognition, we have used YOLOv3[12] Algorithm and pyautogui library. When the Rest API response from the server arrives, the trained model is extracted from it's response body. The python script here runs the webcam infinitely to capture the user's present gesture using the model. After the gesture's class name is detected, the program checks whether the class name is one of the 2 scrolls (Horizontal and Vertical scrolling) or not. If the class name belongs to any of the 4 scrolls, then the user can use his form fist to specify the upward/downward direction or left/right direction and can move his hand with varying speed to control the speed of scrolling of the cursor. To scroll the screen according to the right direction and speed, YOLOv3 algorithm is used to detect the centroid of the form fist of consecutive frames which basically gives us the direction and the distance of the scrolling which needs to be performed. Further, pyautogui library is used to scroll to the calculated distance. In the case where the class name is not one of the 2 scrollings, then directly pyautogui library's function is used to perform the cursor operations.

Parameters	Values
No. of classes	6
Epochs	5
Batch size	8
Learning rate	0.01
Momentum	0.9

Table III
HYPER-PARAMETERS

Algorithm 1: Get trained Model (server side)

Input: Zipped user data
Result: Zipped trained model
Initialize parameters such as LR, batch-size, epochs, etc;
Extract user zip data into a directory;
Define a list of augments with values- blur, contrast, crop, addnoise, etc;
for *frames present in data* **do**
  Initialize 'files' dictionary with key as classname, sample$_no, tfortinaugmentslist$
  for *augtype present in augment list* **do**
    apply augtype to the frame;
    append the frame in a files dictionary corresponding to its sample no (key);
  end
end
for *each key in files dict* **do**
  Extract its classname ;
  Extract list of frames corresponding to each key ;
  Apply spatial transformation on the list of frames ;
  Append preprocessed listoframes and classname as tuple in $training_data$;
end
initialize model with MobileNetV2 object ;
load the state dict of the pretrained model into model ;
change the last (classifier) layer to output 6 values ;
for *parameter, module in model* **do**
  if *module is not classifier* **then**
    append parameter in parameters list;
  else
    append 0 in parameters list;
    break;
  end
end
Make dataloader for $training_data$ with batch size 8;
Initialize optimizer with parameters and momentum ;
Define loss function as cross entropy loss ;
for *epochs in range of 0-5* **do**
  for *inputs, labels in dataloader* **do**
    Extract output from model on this inputs ;
    Find the loss using labels and output ;
    Perform backpropagation step ;
  end
end
return the zipped model

IV. RESULTS

We generated a 9.6 MB model using 30 samples for 6 classes with 90% accuracy. Following are the metrics for the generated model.

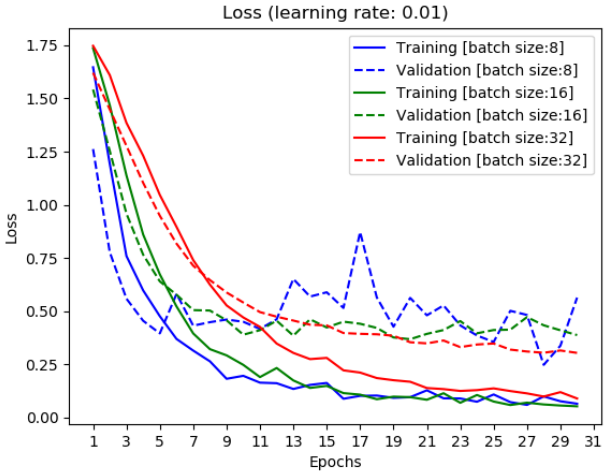

Figure 2. **Training validation loss with LR 0.01**

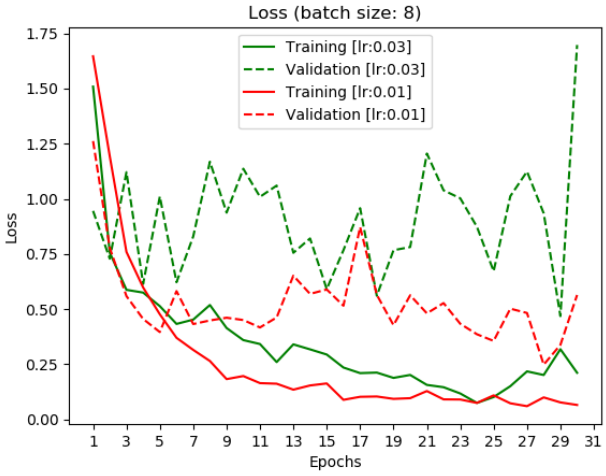

Figure 3. **Training validation loss with batch size 8**

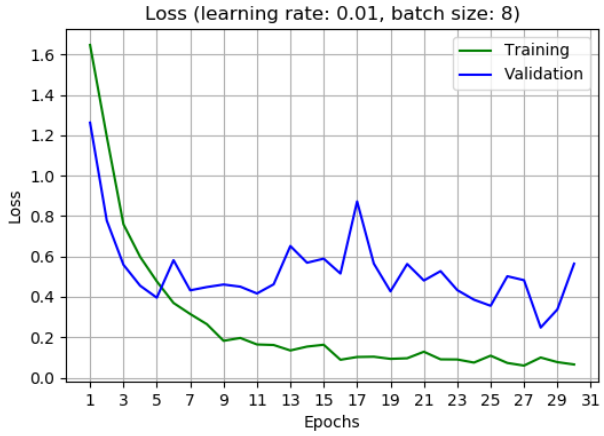

Figure 4. **Training validation loss with LR 0.01 and batch size 8**

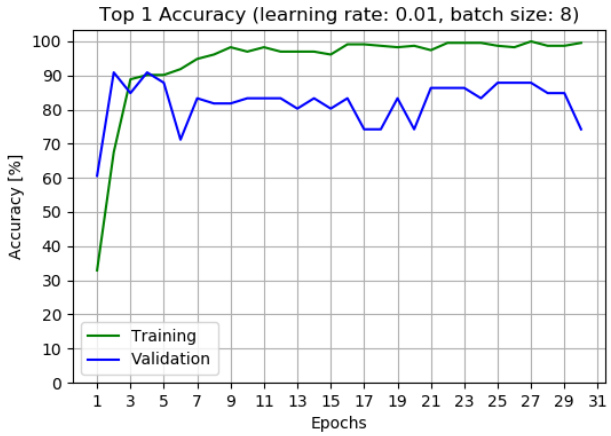

Figure 5. **Training validation accuracy with LR 0.01 and batch size 8**

V. Conclusion

We were able to achieve 90% training accuracy and 88% validation accuracy for our 3D CNN transfer learning model. We were successfully able to integrate our hand detection part with control part. The users can perform scrolling actions, zooming in, zooming out, and changing slides just by their hand gestures. We were successfully able to setup our server and provide the client with a publicly accessible service to training their hand gesture recognition model. In future we plan to work on the GUI on client side, reduce the over-fitting of hand gesture recognition model, and allow more functions like selecting text, copy - pasting it, etc.

References

[1] Okan Kop uklu, Neslihan Kose2, Ahmet Gunduz1, Gerhard Rigoll1 (2019): Resource Efficient 3D Convolutional Neural Networks

[2] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2625–2634, 2015.

[3] Molchanov, P.; Yang, X.; Gupta, S.; Kim, K.; Tyree, S.; Kautz, J. Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D Convolutional Neural Network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 4207–4215.

[4] Jester Dataset: https://20bn.com/datasets/jester/v1

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

[6] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360, 2016.

[7] Kopuklu, Okan Gunduz, Ahmet Köse, Neslihan Rigoll, Gerhard. (2019). Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks.

[8] Okan Köpüklü, Ahmet Gunduz, Neslihan Kose, Gerhard Rigoll, "Online Dynamic Hand Gesture Recognition Including Efficiency Analysis", Biometrics Behavior and Identity Science IEEE Transactions on, vol. 2, no. 2, pp. 85-97, 2020.

[9] Dinh-Son Tran, Ngoc-Huynh Ho, Hyung-Jeong Yang * , Eu-Tteum Baek, Soo-Hyung Kim and Gueesang Lee (2019). Real-Time Hand Gesture Spotting and Recognition Using RGB-D Camera and 3D Convolutional Neural Network.

[10] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, HI, USA, 2001, pp. I-I, doi: 10.1109/CVPR.2001.990517.

[11] S. Rath, "Automatic Mixed Precision Training for Deep Learning using PyTorch", DebuggerCafe, 2020. [Online]. Available: https://debuggercafe.com/automatic-mixed-precision-training-for-deep-learning-using-pytorch/. [Accessed: 01- Nov- 2020].

[12] "cansik/yolo-hand-detection", GitHub, 2020. [Online]. Available: https://github.com/cansik/yolo-hand-detection. [Accessed: 04- Nov- 2020].

VI. Appendix

A. Flowchart

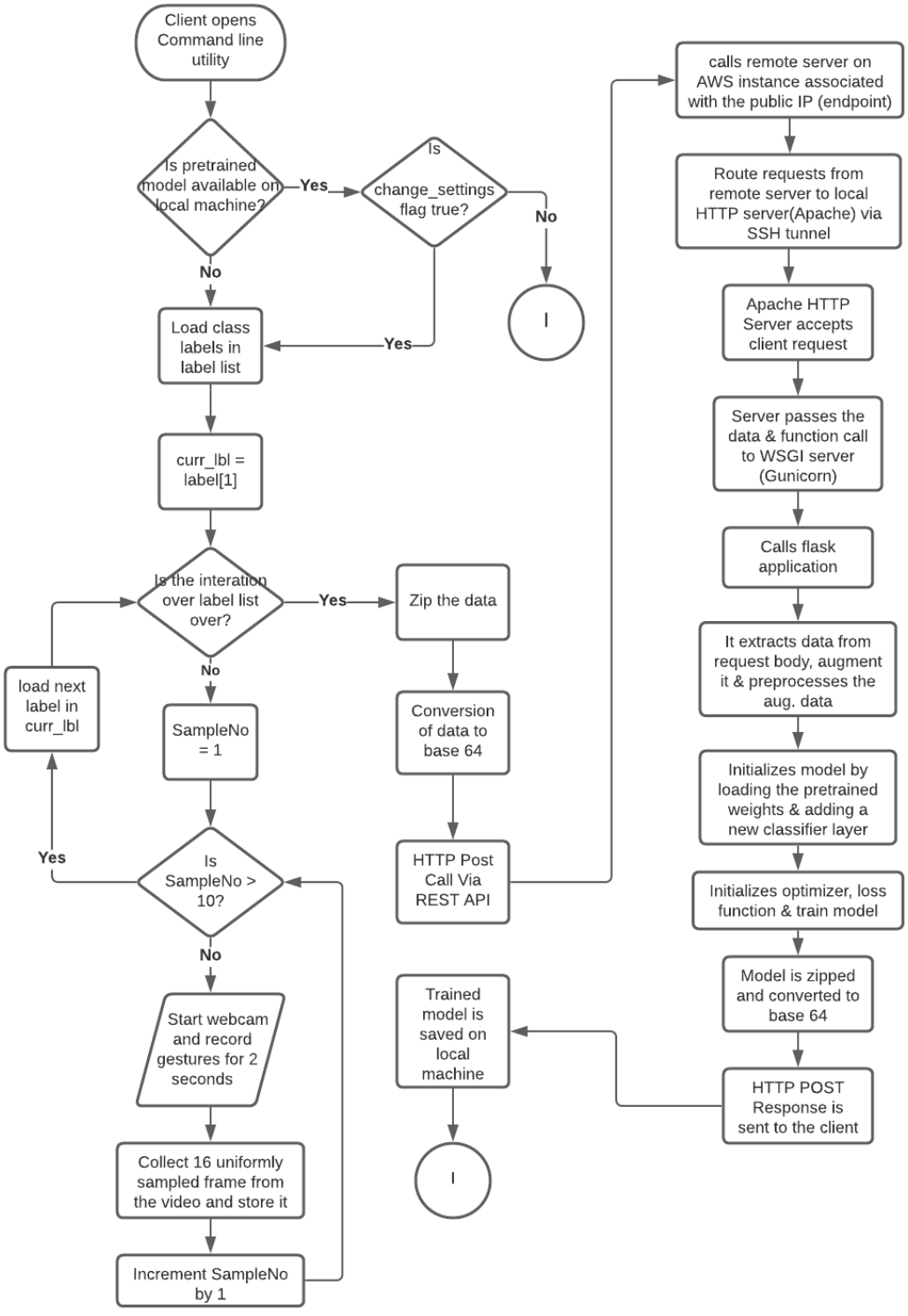

Figure 6. **Data Collection (client) and Server flowchart**

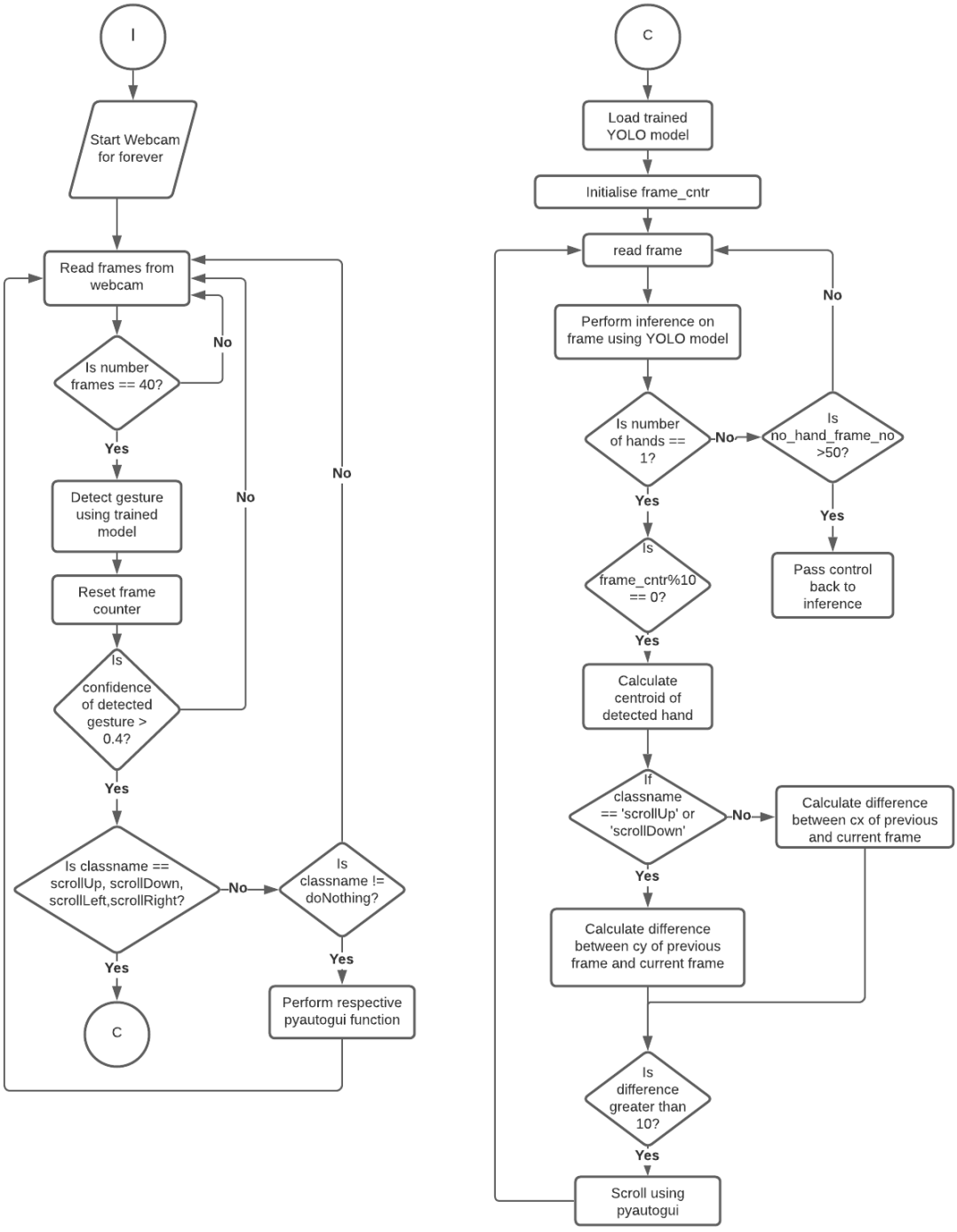

Figure 7. **Inference flowchart (client side)**

B. Sequence Diagram

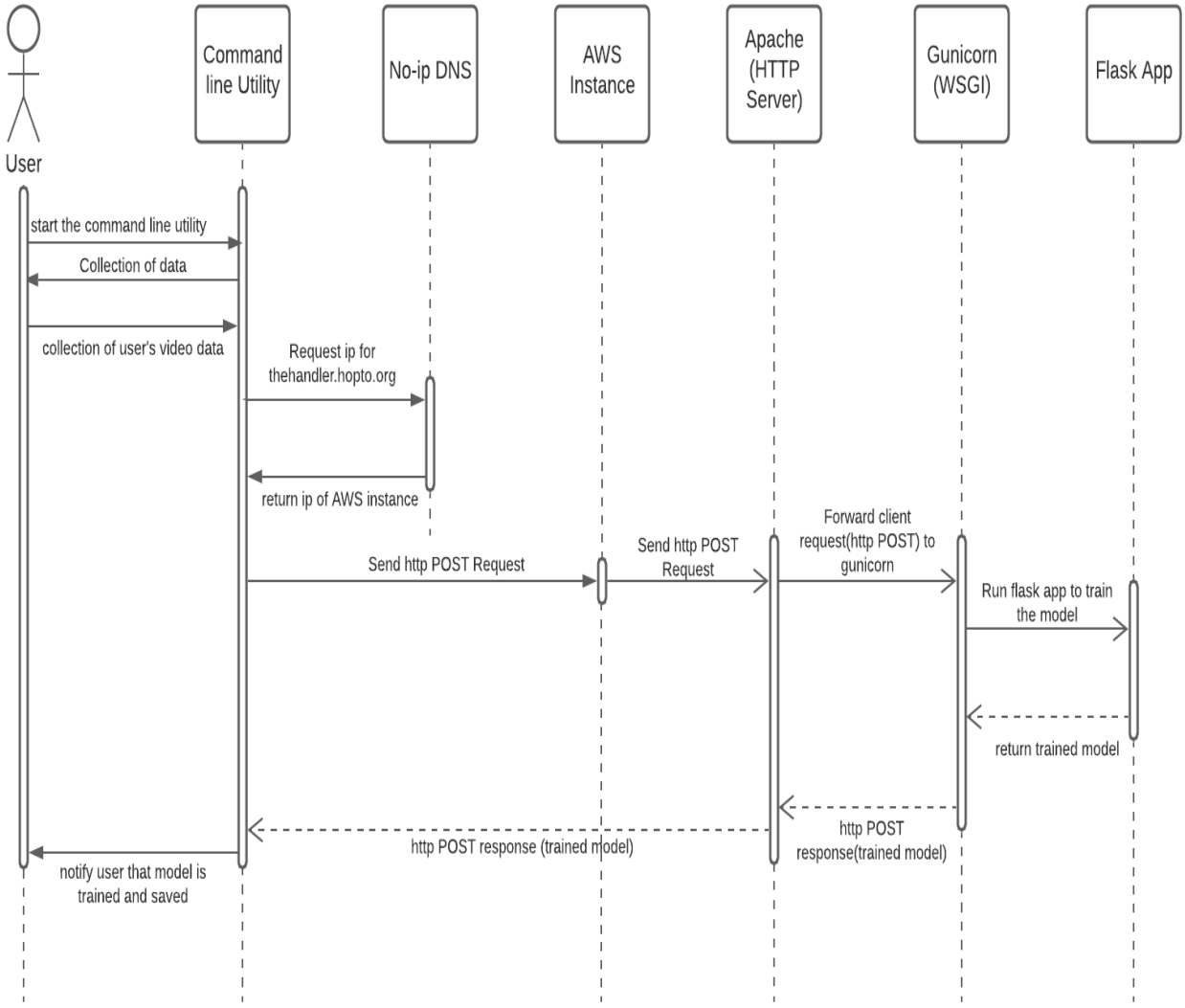

Figure 8. **Sequence Diagram of the system**