Natural Human-Computer Interface Based on Gesture Recognition with YOLO to enhance user experience

Momina Liaqat Ali

I. Abstract

Hand tracking and gesture recognition are rapidly developing fields with many applications in human-computer interaction. This technology enables computers to recognize and respond to hand movements and gestures, creating a more natural and intuitive interface. With the increasing popularity of augmented reality and virtual reality devices, the demand for advanced hand tracking and gesture recognition technologies is growing. The purpose to carry-out this research is to study the current state of the art in hand tracking and gesture recognition and to use the strengths of object detection algorithm for improved techniques for HCI applications with You Only Look Once (YOLO) models that result in the improvement of the user experience in the virtual world.

II. Introduction

A. An overview of Virtual Reality System:

Rapid developments have been occurring in the field of technology. Innovative technologies like Augmented Reality (AR) and Virtual Reality (VR) have seen a sharp increase in demand in the last few years. Demand for virtual environments has surged as a result of the worldwide pandemic, especially the COVID-19 issue, which has sparked a significant change in the field of Artificial Intelligence (AI). These virtual environments have become important because they provide a secure environment in which people may learn new and valuable skills, especially in contexts outside of the conventional classroom.

Isolation became common during the epidemic, and contactless encounters became increasingly important than before. Scientists and researchers from all around the world have

been working nonstop to find solutions for the problems the epidemic has brought about. VR has become a major subject of interest in this setting. Virtual reality VR has become incredibly popular because it provides a special way to address the requirement for smooth interaction. Improving and safeguarding virtual environments is becoming more important as the globe struggles with the effects of a post-pandemic future. This guarantees that in the case of another global health emergency, society will be more equipped and ready.

A common VR setting consists of the following components:

- Real-Time VR Engine
- Software
- Input/ Output devices
- Users
- Tasks

A brief illustration of the flow of a typical VR system in provided in the Fig. 1 below. If we go down further in the VR architecture,

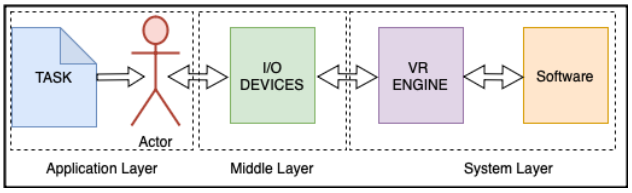

Fig. 1. VR System Architecture

we can divide these five components into three layers to simplify the VR systems. First comes the system layer which is responsible for the tasks of the software and VR Engine and this is the foundational layer. Then comes the middle layer which acts as an interface to connect the user with the VR system and then we have the application layer which includes the tasks and the actors who perform those tasks. These actors can be humans, robots, etc.

B. Challenges in developing VR Systems:

Multiple challenges surround the development of precise and accurate VR Systems. These challenges which are mentioned below demonstrate how difficult it is to create an interface that offers a seamless, intuitive user experience while simultaneously integrating with human motions.

1) Complex Algorithms: Object detection and tracking are popular tasks in AI. These tasks grow increasingly difficult when real-time motion detection across different surroundings is included. They usually need to either build new models from scratch or optimize already-existing ones, both of which require a substantial amount of resources and a large volume of data to be trained effectively.

2) Gesture Recognition and Precision: Accurately capturing human gestures is a significant task. Developing a Gesture Recognition system that can reliably interpret user input is challenging due to the variety of gestures with varying meanings, individual movement patterns, and the requirement for prompt response

all raise the need for a high level of precision in such systems.

3) Real-Time Responsiveness: An additional degree of complexity is introduced by the requirement for real-time responsiveness in virtual settings. To guarantee a seamless and engaging experience, the system must accurately detect and react to human motions instantaneously. The user immersion may be disrupted and the interface efficacy may be reduced by lag or delays in gesture detection, recognition, and response.

4) Integration with various settings: It may be important to extend a system's design to accommodate different uses. Applications are frequently made to be adopted in a particular use case or scenario. Creating a system that is both precise and accurate, yet adaptable enough to be used in a variety of settings is an immense challenge.

5) Cloud Integration: Managing latency and performance is one of the issues of integrating cloud services with VR systems, especially when transferring frames one at a time, which is slow and degrades the performance of real-time systems. Furthermore, the resource-intensive parallel processing of frames is a resource-intensive task and requires significant cloud resources to guarantee seamless and engaging user experiences.

To overcome these barriers, a thorough strategy that incorporates improvements in gesture detection technology, user-centric design concepts, and a thorough comprehension of real-time interaction is required. In this paper, our aim is to use an object detection framework You Only Look Once (YOLO) for gesture recognition and carry out Human-Computer Interaction for accurate and quick recognition of user gestures.

The paper is organized in such a way that in Section 2 we will discuss about the related work done in the field of VR using hand gesture recognition. In Section 3 we will elaborate on the methodology used and Section 5 will discuss the results in detail. In Section 6 and Section 7, we will discuss the applications and analysis of our research. Lastly, we will present the conclusion and future work.

III. Related Work

Scientists have been working on gesture recognition-based tasks for a couple of decades now. Initially, before the advent of advanced AI-based image recognition techniques traditional methods like Markov models, orientation histograms and Finite State Machines (FSM) etc were popular. In the year 2012, when deep learning-based methods were gaining popularity, this field of gesture recognition became extremely popular and now its applications are in numerous fields, among those Medical and

education are the famous areas. The overall flow of the related work section is divided into two main categories:

- Algorithms for gesture recognition.
  1) Traditional gesture recognition techniques
  2) Advanced Deep Learning based gesture recognition
- Hand posture estimation.

A. Algorithms for gesture recognition

1) Traditional gesture recognition techniques: A lot of work has been done in the gesture recognition techniques. Even before the popularity of deep learning, scientists have been working to improve the already existing or creating new advanced techniques to recognize different gestures. Chen et al [2] used hidden Markov models to recognize different hand gestures. They tracked the hand in real-time and used Fourier Descriptor and motion analysis for characterizing spatial features and temporal features respectively. After characterizing these features they combined them to form a feature vector for the given sequence of input images and then applied hidden Markov Models for gesture recognition. They recognized different gestures with an overall recognition rate of more than 90%.

In another work by Freeman et al. [3], they used an orientation histogram for gesture classification and recognition. They first transformed the image into a feature vector by using a histogram of local orientation and then compared it with the available feature vectors of gestures available in the training set and for this comparison, they used Euclidean distance. In the case of dynamic gestures or moving hands, the spatio-temporal gradients were utilized as feature vectors. Their approach was simple as well as efficient for gesture recognition tasks. Hong et. al. used the concept of the finite state machine to propose a new algorithm in which for efficient gesture recognition they not only included the hand but also the center of the user head. They used spatial clustering these clusters were formed into segments along with temporal information about the gestures which were further used for building FSM recognizer [4].

2) Advanced Deep Learning based gesture recognition: Ozdemir et. al. utilized the potential of deep learning for accurate hand gesture recognition. They monitored the Electromyography (EMG) signal by placing 4 surface EMG electrodes on user arm. Furthermore, they applied Short-Time FT on the obtained spectrogram images from the observed sEMG signal. These spectrogram images were passed to a CNN to classify the hand gestures based on 7 distinct gestures [9]. YOLO-based models are popular for gesture recognition since they per-

form reliable object detection and are single-stage algorithms which makes them suitable for real-time applications [7][10][5] . Yu et al. used YOLOv3 for this task. They used dedicated depth cameras to take hand images and used depth information to train the YOLOv3 for hand movement recognition [14]. Similarly, Artificial Neural networks (ANNs) have also been used for gesture recognition [1]. Transfer learning is becoming popular in Deep Learning (DL) tasks since DL models are data-hungry and they require a huge amount of data to train on. For this purpose, per-trained models come to our help. Savas et. al. used multiple two-stage pre-trained models and evaluated their performance on HG14 hand data [11]. Similarly, Ozedemir et. al. used Time Frequency (TF) images along with seven distinct CNN architectures which were all pre-trained models and performed the classification of seven hand gestures. Then they used stratified and leave-one-out cross-validation techniques to evaluate their proposed model [8].

B. Hand posture estimation

Various studies have been conducted to optimize posture estimation using deep learning techniques. Wang et al. conducted experiments to improve the accuracy of recognizing hand gestures using sEMG. Although this technique involves the placement of electrodes on a human arm they were able to achieve a higher accuracy. They combined CNN with transfer learning to strengthen the overall system to perform better for new users [13]. In another study, Tang et. al. used kinetic sensors and Deep Neural Networks (DNN) to perform posture estimation using normal cameras. They divided their algorithm into two distinct stages where in the first stage they combine color and depth information to find the hand in the image and in the next stage they feed these images to DNN to recognize the gesture [12]. When dealing with tasks where we have large amounts of data and need to attain higher accuracy as well a higher need for huge computational resources arises. Bonab et al. dealt with this issue and proposed two hybrid DNNs for accurate posture estimation. They reduced the dimensionality of the data while data travels within the DNN network but to cope with information loss they used residual connections to retain some of the information. This way fewer computational resources were required [6].

IV. Methodology

A. Dataset

To address the hand gesture recognition task, we created a customized dataset with over 20,000 images. Each image was taken with careful consideration for the background, light-

ing, and hand position. Different poses of both hands were displayed to the camera. Each image is 2666x1488 pixels and was captured with a typical webcam against a green backdrop. To increase the size and to bring variation in data we employed a couple of augmentation techniques like grayscale conversion and flipping. The overall construction of the dataset can be seen in Fig. 2.

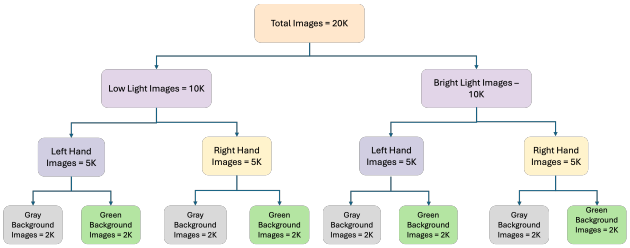

Fig. 2. Dataset Construction Tree

1) Annotating the dataset: In our hand data, we aimed to locate 21 key points or landmarks on a human hand. These landmarks represent the joints and tips of fingers and one dedicated landmark for the wrist. The landmarks and their positions and IDs can be seen in Fig. 3 below. To annotate the dataset typically

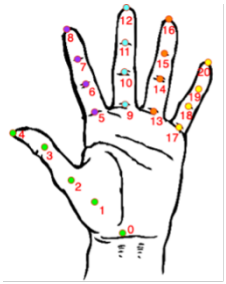

Fig. 3. Landmarks used to annotate Hand Data

two approaches are commonly used which are discussed below:

- Manual Annotations
- Automated Annotations

Manual Annotation: Initially, we began with manual annotations, we used an open-source tool for locating and marking 21 landmarks on each human hand. This task was laborious and needed precision. In the cases where the dataset is large, manual annotation becomes an impractical solution. Moreover, human subjectivity can be an important factor in affecting the consistency of manual annotation tasks since fatigue, level of expertise and human individuality can greatly impact the annotations. These imprecise annotations or errors when propagated to the training process can lead to undesired results and training such models over and over again will excessively utilize the resources without improving the model accuracy.

Automated Annotation: For Automated annotations, we used a popular library called MMPose which is a part of MMLab Framework which is an open-source toolkit developed in PyTorch. For hand data annotation, MMPose uses Real-Time Object Detectors (RTMDet) which is a real-time object detector. The architecture of RTMDet comprises of Backbone for feature extraction, a Neck for enhancing the feature space, and a Head to make accurate final predictions. It is trained on 4 diverse hand

datasets which are OneHand10K, FreiHand, RHD(Rendered Hand Dataset) and HalpeHand. We used the trained RTMDet-nano model for our particular application to reliably identify hands in our dataset. After the detection stage, we used another model from the MMPose suite called RTMPose to carry out detailed posture estimation, which performs the tasks of estimating the exact locations of hand key points. We were able to annotate our dataset successfully and efficiently by combining both RTMDet for detection and RTMPose for posture estimation. This substantially reduced the amount of human effort needed for this laborious task. The sample annotated images we obtained after using MMPose are shown in Fig. 4 below.

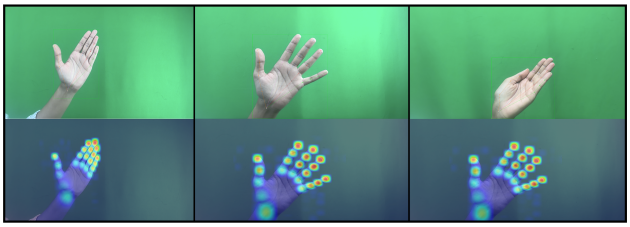

Fig. 4. Annotations using MMPose

B. Hand Gesture Recognition using YOLO

YOLO is a powerful framework that is commonly used to perform real-time object detection in various applications. Its ability to perform detection in a single pass makes it superior to two-stage object detectors. Although two-stage detectors are considered more accurate since they divide the detection task into two stages: i) They first do Region Of Interest (ROI) detection. ii) they perform localization on that region. These detectors are slow and commonly used in tasks where high accuracy is required like in the medical field. Contrary to these detectors, single-stage detectors perform both the ROI detection and localization in a single pass which makes it fast especially for real-time applications.

For hand detection, we used a pre-trained YOLO model which is YOLONAS-Pose for hand posture estimation and fine-tuned it on our custom dataset. Yolo-NAS Pose model is trained on a huge COCO2017 dataset. This large-scale dataset includes more than 200,000 unique instances of human postures, each labeled with 17 landmark points. We used a fine-tuning approach to improve the performance of the model as deep learning models require a large amount of data for training. We fine-tuned the model on our custom dataset by using its already gained knowledge from pre-training on the COCO2017 dataset. Fig 4 shows how carefully each landmark of the human hand is annotated with 21 essential key points which correspond to the finger joints and wrist of a human hand. By using fine-tuning as an essential adaptation mechanism, the model can focus on precisely identifying and tracking hand postures according to the distinct features in our

hand dataset. This method not only reduces the data scarcity problem and refines the model, but also makes it capable of providing accurate hand posture estimation that can be helpful in a variety of real-world applications.

To make our data compatible with the YOLONAS Pose model, we resized the images and maintained the aspect ratio by using the equations (1),(2) and (3):

$$r = max(\frac{640}{h}, \frac{640}{w}) \tag{1}$$

$$updated\_height = h * r \tag{2}$$

$$updated\_width = w * r \tag{3}$$

Where **r** is the scaling factor **h** is the height of the image and **w** is the width of the image. We resized the image based on the scaling factor. Since we have changed the height and width of the image the key-points and bounding box coordinates must be updated as well. To update them, we used the equations (4) and (5):

$$updatedKeypoint_i = originalKeypoint_i * r \tag{4}$$

$$updatedBoundingbox_i = originalBoundingbox_i * r \tag{5}$$

The overall hand gesture recognition task was divided into three stages Hand Recognition, Hand Tracking, and Gesture Recognition.

1) Hand Recognition: For hand recognition, we have used a regular webcam which takes live frames as input and detects a human hand in each frame. For hand detection, YOLONAS Pose is employed which not only performs detection but also does key-point localization in real time. This is the first and most important step in our overall hand recognition task.

2) Hand Tracking: The next important step is to follow the hand movement once it has been detected in the video frame. The ability to monitor the hand movements continuously across several frames is made possible by YOLO grid division and bounding box prediction. The system maintains a precise grasp of the hand position in real-time by continuously updating the bounding box coordinates, which guarantees smooth and accurate tracking. This stage allows the system to respond to the user gestures in real-time by creating the link between the first-hand recognition and the subsequent gesture recognition phase.

3) Gesture Recognition: The last step is identifying certain hand motions while the hand movement is being monitored. Every motion, including an extended palm, closed fingers, an open hand, etc performs certain kind of functionality. Real-time processing of YOLO allows the system to identify movements accurately and ignite the corresponding functionality associated with the corresponding gesture in real-

time. In our case, certain key points are being monitored which have IDs 8 and 12 which represent the tips of the index and middle finger respectively. The distance between these key points is being monitored using Euclidean distance and if the distance goes below the threshold we initiate the operation the equation used can be seen in (6). This feature enables a clear interpretation of hand motions, resulting in a natural and easy-to-use interface for hand-based control applications or virtual environments.

$$distance = \sqrt{(x_1 2 - x_8)^2 + (y_1 2 - y_8)^2} \quad (6)$$

The above-described three-stage pipeline of gesture recognition which uses YOLONAS capabilities is illustrated in Fig. 5 below.

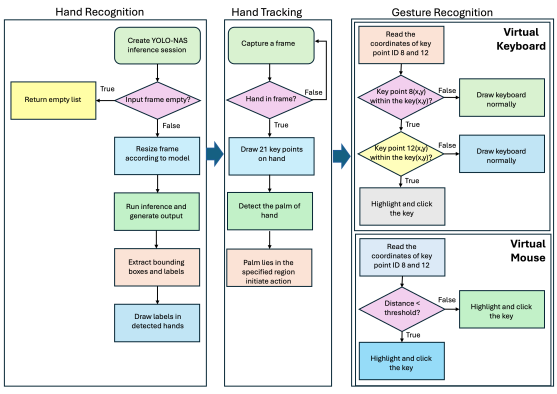

Fig. 5. Three Step process of hand gesture recognition task

C. Natural HCI Design

For implementing the virtual mouse and keyboard on a local system we used OpenCV. OpenCV is a Python library and it is useful for developing time-responsive computer vision applications.

1) Virtual Mouse: In our study, we used the weights of our object detection model to identify and track the user hand. A typical webcam available on laptops is used as an input device to capture and record frames. The program analyzes these recorded frames and determines the position of the cursor on the computer screen based on hand movement. PyAutoGUI is a well-known Python library that supports various automation tasks and assists in replicating mouse and keyboard functionalities.

When a hand is detected in the frame, the algorithm uses the mean key-point score to determine the centroid of the palm. If the palm centroid is detected with the pre-drawn rectangular region on the screen, then the gesture control function is triggered. The mouse pointer is then moved to the middle of the screen using PyAutoGUI. Now moving the hand can move the cursor on the screen. If there is no hand in the camera view the gesture control operation will be disabled. The key-point 8 which is the tip of the index finger is responsible for rotating the mouse virtually. Key-point 12 is the ID assigned to the tip of the middle finger. These IDs can be seen in Figure 3 in detail. If the distance between the key points 8 and 12 is significantly small, then a click operation is

initiated. The overall workflow of the virtual mouse is shown in Fig. 6 below:

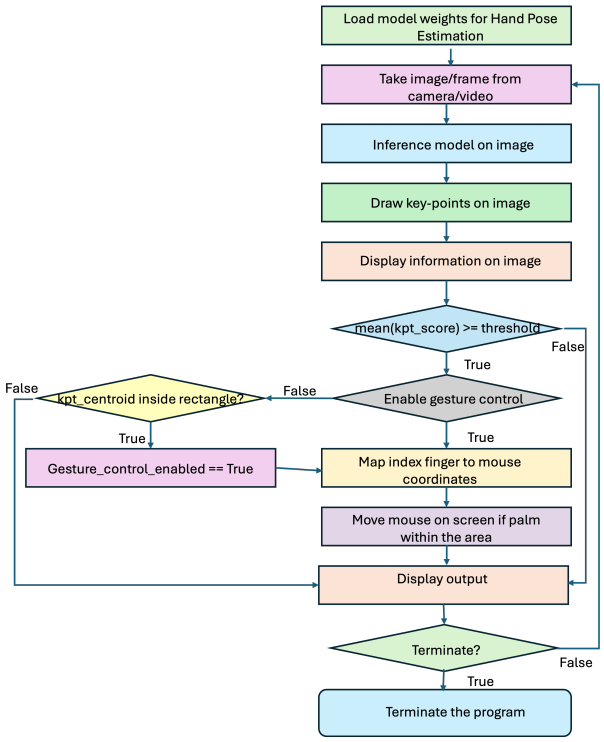

Fig. 6. Detailed workflow of virtual mouse for computer system

2) Virtual Keyboard: We implemented a virtual keyboard with an intuitive user interface displayed on the screen. To record user input, an empty list of characters is created when the application is launched. Furthermore, to detect if a button on the virtual keyboard is pressed the tracking system tracks the middle and index fingertips, designated as key points 8 and 12, respectively. The overall logic of how a key hit is detected is briefly explained in the algorithm below:

- Based on the keyboard layout, determine the size and position of the keys on the on-screen keyboard.
- Check if the fingertips of the index and middle finger are within the boundary of the key.
- Highlight the key if the middle and index finger have their tips in key boundary and update the text on the screen. If the key is 'BK' then delete a character from the list of input characters.
- If only the index finger is on the key just highlight that key.
- If none of these two fingertips are on a key, then draw keyboard normally.

Fig. 7 shows the detailed flow of our virtual keyboard interface. First, the coordinates of index finger are computed and if they are within any key of the on-screen keyboard then we check the coordinates of the middle fingertip. If both the x-axis and y-axis of the fingertips and keyboard buttons are within the same range the user receives visual feedback as the input key on the virtual keyboard becomes highlighted and is considered pressed. Additionally, as new keys are pressed, the list of input keys is constantly updated. With this interactive design, users may type characters with ease by using the natural motion of their fingers.

V. Results

We performed experiments on two variants of the dataset. One contains 5k images - dataset

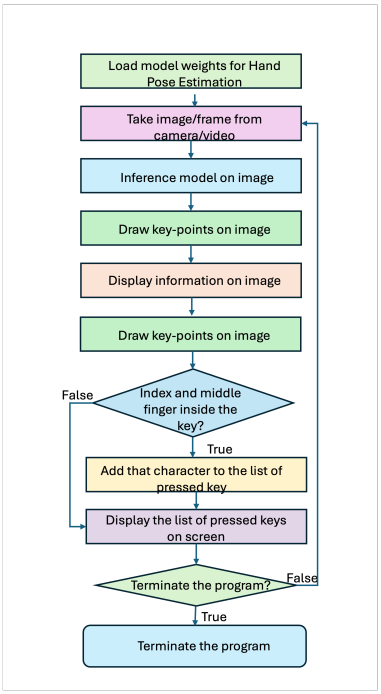

Fig. 7. Detailed workflow of virtual keyboard for computer system

A, while the other variant contains 20k images - dataset B. Dataset A contains images with green background while in Dataset B same amount of grey background images were added using augmentation. TABLE I highlights the different values chosen for hyperparameters. For dataset A we made the split of 80-20 with

Configuration	Value
Epochs	10
Learning Rate	0.001
Optimizer	AdamW
Weight Decay	0.000001
Batch size	32
Iterations Per Epoch	439 (len(train_loader))

TABLE I
DETAILS OF HYPERPARAMETERS FOR MODEL TRAINING.

80 for training and 20 for testing while in dataset B we did 70-30 splitting. The image size of 640*640 was fed to the model. To carry out all the experiments, we used a cloud-based L4 GPU, and the specifications of this GPU are listed in TABEL II

Features	Details
CUDA Cores	7680 Cores
GPU Memory	24 GB GDDR6 @ 300 GBps
Compute Performance FP64	0.5 TFLOPS
Compute Performance FP32	30.3 TFLOPS
Architecture	NVIDIA Ada Lovelace

TABLE II
A SCALED-DOWN TABLE.

We implemented Datasets A and B on three different variants of YOLONas-Pose which are nano, small, and medium. These models differ based on the number of parameters and the total size of the model. For example, the total number of parameters used by YOLONAS-Pose in the nano variant are 9,901,483 which increases in the small variant and reaches a whooping figure of 22.2 million and these parameters reach almost 58.2 million in the medium variant. For comparison purposes, each model was trained for 10 epochs and we analyzed different losses and Average Precision and Recall of each model. We monitored classification loss to see how well the model can classify if there is an object in the grid box while Intersection Over Union (IOU) loss is used to calculate the actual difference between the prediction made by the model and the actual ground truth. It is good to compute IOU loss when we want to know how closely the model is performing

to actual labels. And lastly, we monitored the total loss on each epoch. The three Loss curves IOU Loss, Classification Loss and total Loss for Nano model on Dataset A for Train and Validation sets can be seen in Fig. 8 below.

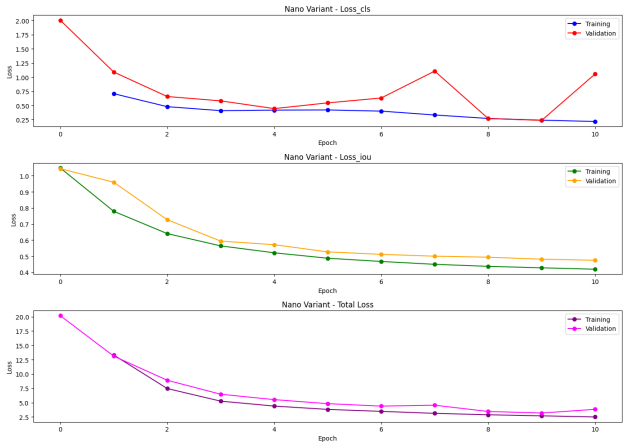

Fig. 8. Three Different Loss Analysis on YOLONAS-Pose nano variant using Dataset A

Similarly, the same loss metrics were computed for small and medium variants of YOLONAS-Pose as well between Train and Validation splits. These comparison graphs can be seen in Fig. 9 and 10 respectively. Similarly,

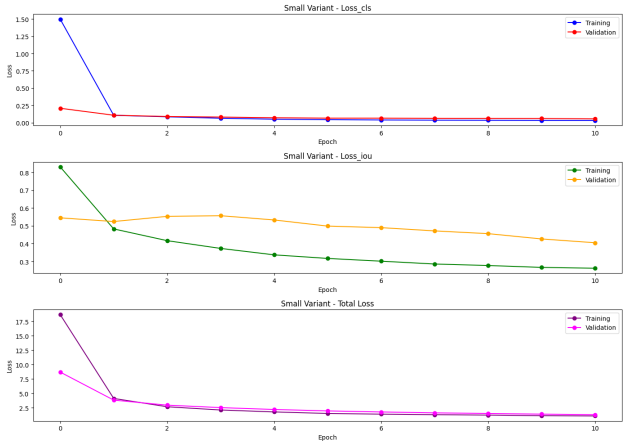

Fig. 9. Three Different loss analysis on YOLONAS-Pose small variant using Dataset A

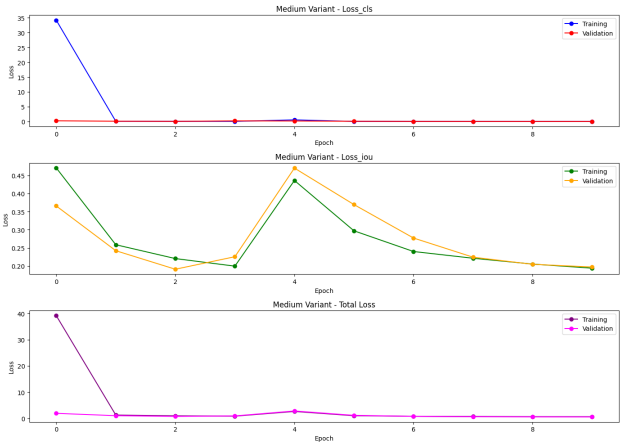

Fig. 10. Three Different loss analysis on YOLONAS-Pose medium variant using Dataset A

for Dataset B same YOLONas-Pose variants were tested and the loss curves for the three variants are shown in Fig. 11, 12 and 13.

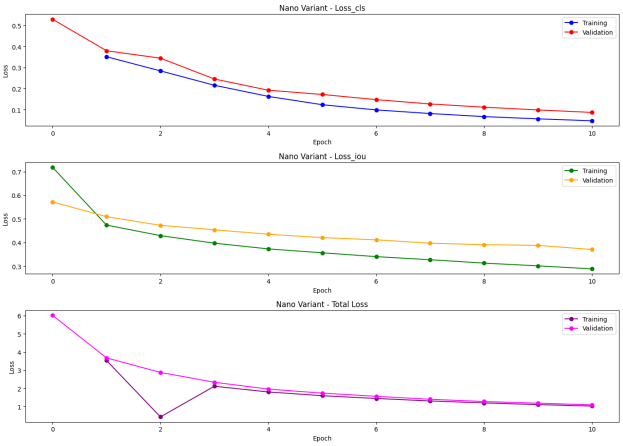

Fig. 11. Three Different loss analysis on YOLONAS-Pose nano variant using Dataset A

The bar chart of AP, AR and F1 on two variants of our dataset are shown in Fig. 14 and 15 below. Moreover, we imported the best weights to our OpenCV application and implemented a virtual mouse and keyboard. The high precision of the model gave us good key-

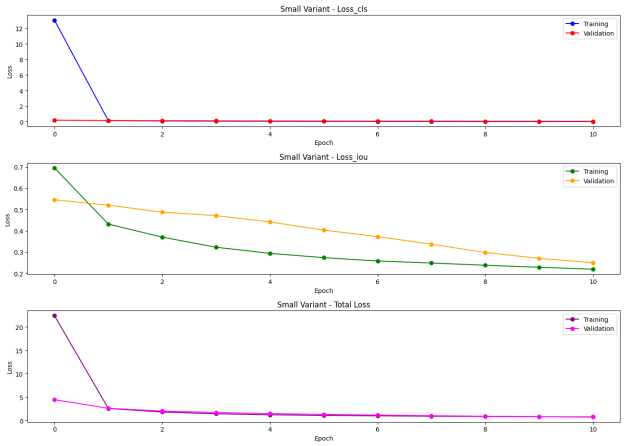

Fig. 12. Three Different loss analysis on YOLONAS-Pose small variant using Dataset A

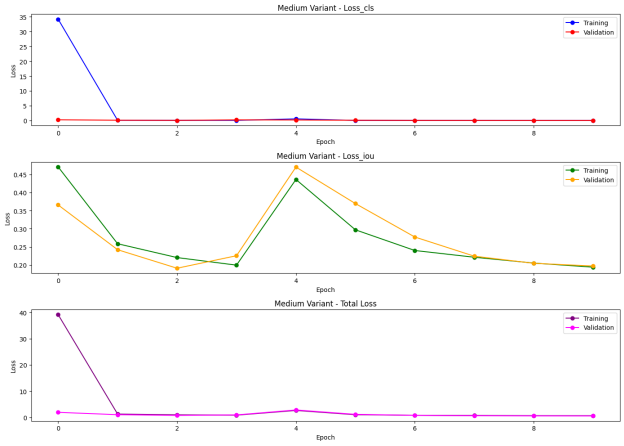

Fig. 13. Three Different loss analysis on YOLONAS-Pose medium variant using Dataset A

We also observed the AP, AR and F1 scores of these models. The results for datasets A and B are listed in Table III and IV respectively.

Model Variant	AP	AR	F1
Nano	0.9886	0.9921	0.9903
Small	0.9883	0.9931	0.9906
Medium	0.9886	0.9921	0.9903

TABLE III
RESULTS OBTAINED ON DATASET A

point detection on our real-time application. We tested the application with green background and Figure 16 and 17 show the instances of

Model Variant	AP	AR	F1
Nano	0.9890	0.9959	0.9931
Small	0.9896	0.9962	0.9928
Medium	0.9960	0.9995	0.9977

TABLE IV
RESULTS OBTAINED ON DATASET B

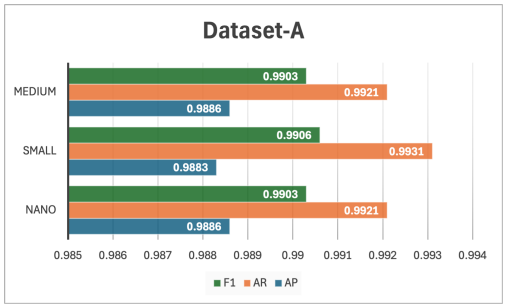

Fig. 14. Bar chart of observed metrics for Dataset A

our virtual mouse and keyboard respectively. Both virtual mouse and keyboard take real-time frames, process the frames, and give us a frame rate of 6 Frames Per Second (FPS). Seeing the results we obtained on both datasets, we decided to make Nano model as our final prediction model to be used to implement a virtual mouse and keyboard. Since the nano model is a lightweight model and can be used effectively on edge devices. We trained it for 50 epochs and seeing the process of the training

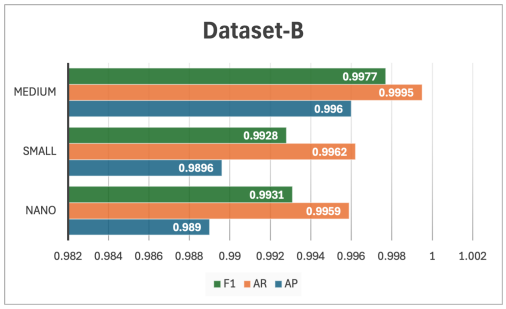

Fig. 15. Bar chart of observed metrics for Dataset B

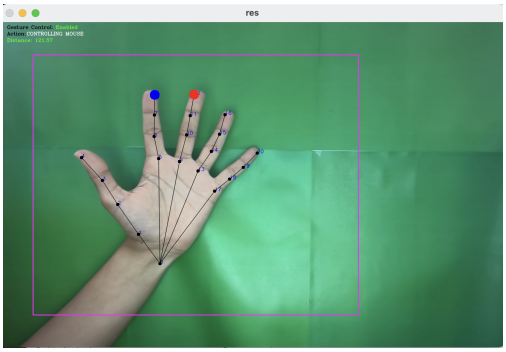

Fig. 16. Instance of our virtual mouse

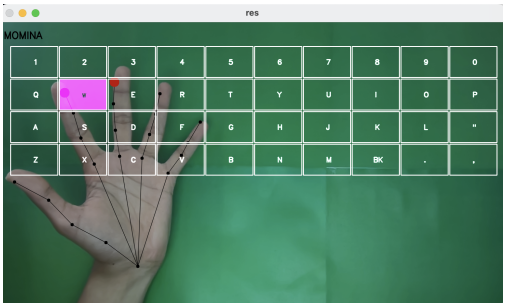

Fig. 17. Instance of our virtual keyboard

epoch by epoch we can see that model is not quickly going towards overfitting. The results can be seen in Fig. 18 and 19.

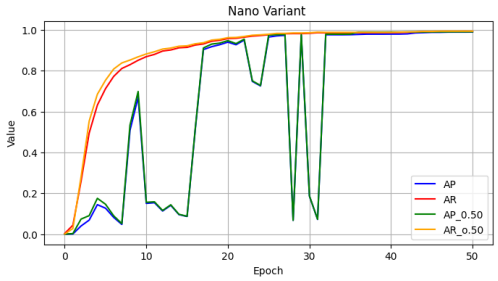

Fig. 18. AP and AR curves over 50 epochs

VI. Applications

The virtual keyboard and mouse are a ground-breaking development in user interface design that has the potential to greatly improve

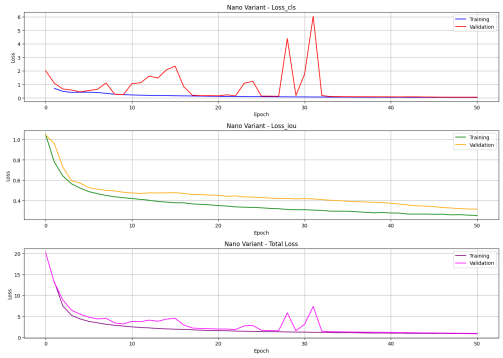

Fig. 19. Comparison of loss curves for nano model

the gaming and education sectors. Although people have traditionally interacted with computers primarily through mouse and keyboard inputs, the introduction of virtual input technologies adds a new level of involvement that has the potential to revolutionize these disciplines. Virtual mouse and keyboards provide an alternative to real lab equipment in distant learning contexts where physical access may be limited. Through the use of computer screens, they offer an engaging experience that lets students operate devices and conduct virtual experiments. By simulating real-world interactions, these technologies improve access to high-quality educational resources and allow for remote, hands-on learning.

Also, Virtual input devices provide great potential for innovation in a gaming business. More realistic and user-friendly controls that surpass conventional limitations of hardware might be advantageous to gamers. More realistic and captivating gaming is possible when vir-

tual mouse and keyboards are incorporated into augmented reality (AR) and virtual reality (VR) settings. With the use of these technologies, gamers may interact with virtual environments in ways that are more like human actions and gestures in our real world, allowing entertaining experience to users. To demonstrate the use of our virtual mouse we tested it on a Chess game on a computer Fig.20. Also, we used it as a virtual remote control for media for pausing and playing the video on computer Fig.21.

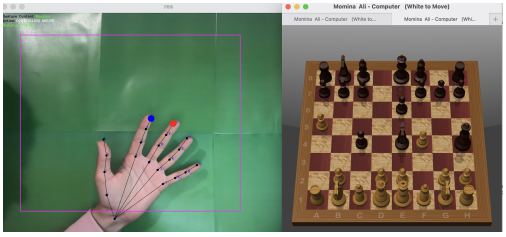

Fig. 20. Instance of playing Chess Game on a computer using virtual mouse

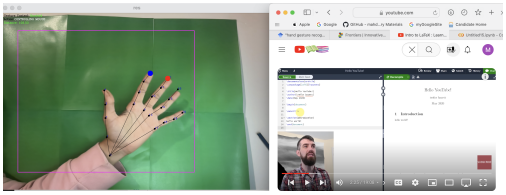

Fig. 21. Instance of controlling media on a computer using virtual mouse

VII. Analysis

We tested the models on two variants of the dataset. One variant has around 5K images while the other one has 20K images. If we see the results presented in section VI we can see that these models are achieving very high precision and recall of almost 99% on 10 epochs. This shows that there is a clear chance of overfitting in all those models. Multiple combinations of hyper-parameters were tested to eliminate this problem but we realized that the issue is with the dataset itself. The variant of dataset which we have named as Dataset A which contains 5K images has all the images with green background. Although the lighting conditions are not static but all the images are taken with the same green background. So, we tried to eradicate this problem and augmented the data and flipped the images and also added images that have grayscale augmentation applied to them. This way we not only increased the data but also brought some level of variation in the dataset. However, the problem of overfitting persisted. THe effect of this problem becomes evident when we test our virtual mouse or keyboard with some other color of background or if the human hand changes. In other words, we can say that the model is highly subjective to one individual. Moreover, the keypoint detection in real-time application is really improved and when we test the model with green background model never fails to detect the hand and locate the 21 keypoints with great accuracy. Also, the framerate is decreased to only 6 frames per second and that is probably because the model weights are coming from a complex model and

on real-time application model requires time to process each frame. Also, the framerate on both flask and uWSGI servers is also very small.

VIII. Conclusion and Future Work

Our research focuses on the HCI in Virtual environments and the aim is to provide user with immersive experience to carryout different tasks while dealing with computers in contact-less setting. We demonstrated the use cases of our model in both gaming and education section by playing a computer game and designing a simple circuit for simulation purposes. The strengths of YOLO were utilized as the object detector to provide high level of precision in real-time object detection tasks.

Moreover, in future, we want to work on making our dataset diverse by adding more images with different backgrounds and by adding noise as well so the model can have tough time in detecting the key-points on human hand. Also, more images will be taken by having a varying distance from the camera since changing the distance in our current application is also having some degree of effect in real-time performance. Cloud interface will be made better by first improving the model ability to process frames rapidly and then we will prioritize the use of uWSGI so parallel processing of the frames can be achieved. Currently, two gestures are implemented like click and move. In future we will include the drag feature as well.

IX. Acknowledgements

This work has been conducted at MTSU by a first year PhD candidate in Computational and Data Science porgram. We want to thank our supervisor Dr Zhou Zhang for guiding us during the process. Also we want to thank the Director of the program Dr John Wallin for giving us an opportunity to compile the overall work and guide us on making our projects better. We want to thank the staff at the James E. Walker library for their cooperation for letting us use the green screen room in maker's space to collect the dataset.

References

[1] Vinod Adithya, PR Vinod, and Usha Gopalakrishnan. Artificial neural network based method for indian sign language recognition. In *2013 IEEE conference on information & communication technologies*, pages 1080–1085. Ieee, 2013.

[2] Feng-Sheng Chen, Chih-Ming Fu, and Chung-Lin Huang. Hand gesture recognition using a real-time tracking method and hidden markov models. *Image and vision computing*, 21(8):745–758, 2003.

[3] William T Freeman and Michal Roth. Orientation histograms for hand gesture recognition. In *International workshop on automatic face and gesture recognition*, volume 12, pages 296–301. Citeseer, 1995.

[4] Pengyu Hong, Matthew Turk, and Thomas S Huang. Gesture modeling and recognition using finite state machines. In *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*, pages 410–415. IEEE, 2000.

[5] Soukaina Chraa Mesbahi, Mohamed Adnane Mahraz, Jamal Riffi, and Hamid Tairi. Hand gesture recognition based on various deep learning yolo models. *International Journal of Advanced Computer Science and Applications*, 14(4), 2023.

[6] M. Mofarreh-Bonab, H. Seyedarabi, B. Mozaffari Tazehkand, et al. 3d hand pose estimation using rgbd images and hybrid deep learning networks. *The Visual Computer*, 38:2023–2032, 2022.

[7] Abdullah Mujahid, Mazhar Javed Awan, Awais Yasin, Mazin Abed Mohammed, Robertas Damaševičius, Rytis Maskeliūnas, and Karrar Hameed Abdulkareem. Real-time hand gesture recognition based on deep learning yolov3 model. *Applied Sciences*, 11(9):4164, 2021.

[8] Mehmet Akif Ozdemir, Deniz Hande Kisa, Onan Guren, and Aydin Akan. Hand gesture classification using time–frequency images and transfer learning based on cnn. *Biomedical Signal Processing and Control*, 77:103787, 2022.

[9] Mehmet Akif Ozdemir, Deniz Hande Kisa, Onan Guren, Aytug Onan, and Aydin Akan. Emg based hand gesture recognition using deep learning. In *2020 Medical Technologies Congress (TIPTEKNO)*, pages 1–4. IEEE, 2020.

[10] Yue Pan and Ran Ren. Research on gesture recognition based on neural network and virtual reality technology. In *2024 4th International Conference on Neural Networks, Information and Communication (NNICE)*, pages 1648–1652. IEEE, 2024.

[11] Serkan Savaş and Atilla Ergüzen. Hand gesture recognition with two stage approach using transfer learning and deep ensemble learning. *arXiv preprint arXiv:2309.11610*, 2023.

[12] Ao Tang, Ke Lu, Yufei Wang, Jie Huang, and Houqiang Li. A real-time hand posture recognition system using deep neural networks. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(2):1–23, 2015.

[13] Yanyu Wang, Pengfei Zhao, and Zhen Zhang. A deep learning approach using attention mechanism and transfer learning for electromyographic hand gesture estimation. *Expert Systems with Applications*, 234:121055, 2023.

[14] Xiaoyang Yu, Lin Jiang, and Lijun Wang. Virtual reality gesture recognition based on depth information. In *SID Symposium Digest of Technical Papers*, volume 51, pages 196–200. Wiley Online Library, 2020.