Gesture Controlled UI/UX Navigation using Neural Networks

Rajat Bisht
Khoury College, Northeastern University
bisht.r@northeastern.edu

Abstract

Gesture controlled UI/UX implementation in this Project[10] leverages open-source mediapipe-hands[2] library from Google to develop a gesture recognition engine that tracks and synchronizes mouse operations. The model trained showed 94% accuracy on testing data while finishing gesture classification operation in an average of 0.000819 seconds while with mediapipe model takes 0.03 seconds.

1 Introduction

Hand gesture recognition using computer vision has gained significant importance with the emergence of new Human-Computer Interaction (HCI) technologies in form of Virtual Reality(VR), Augmented Reality(AR), touchless user-interfaces, and assistive technologies. Hand gestures play a crucial role in navigation within these meta-realities, requiring development of gesture classification and recognition piplines/engines, with little latency and higher throughput for a faster, seamless, and highly accurate mapping. Through this project, I have created an engine that can simulate a mouse pointer UI/UX using hand gesture capable of platform independent deployment.

I felt motivated to work on this particular project because of a lack of comfort I kept encountering while watching movies, reading a research paper, or a book on a big projector screen. I found myself moving to-and-fro between my chair and my laptop. Therefore I decided to utilize my knowledge from course 'CS5330 Pattern Recognition and Computer Vision' to create a software solution that can bridge this gap of accessibility technology.

This project[10] is very loosely based on sixth sense technology [1] developed by Pranav Mistry in association with MIT Media labs. Building upon the idea, this project implements a Neural Network based approach for optimal and faster hand-gesture recognition using webcam feed. The output of this trained neural net is used directly for HCI application by programmatically implementing mouse operations, for mouse-pointer navigation, mouse button-clicks and scrolling-wheel. This offers users of this application to be able to interact with their desktop or laptop devices without requiring a need of physical touch.

This project[10] draws a contrast from other two previously implemented and academically accepted approaches that formed a basis of continued research in the field of gesture recognition and its transference to the field of Human Computer Interface (HCI). First is a traditional image processing based approach, where authors implemented various feature extraction algorithms in MATLAB scripting environment to successfully classify 39 different gestures form still images[3]. Second is a continuation on the same tangent with inclusion of machine learning based algorithms[4] where authors used some traditional feature extraction mechanisms to crop out hand features and used a temporal 3D convolutional neural net to classify gestures and map them to mouse operations.

This project[10] focuses on real-time gesture recognition system that detects fingers' position, tracking their movement and classifying the gestures into one of the seven predefined categories. For implementing this pipeline, I have leveraged highly efficient and open-source Google Mediapipe

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

project [2] for palm detection and hand-landmark extraction, which is then fed into a Feed Forward Network that classifies each detected gesture. For the Gestures that map to a navigation dependent operation like mouse pointer movement and scrolling window, position of index finger's tip is used as guide for a smoth synchronizing movement. For the scope of this project, 'left' and 'right' handedness is not considered symmetrically. I have trained the model to learn, map and respond outputs only for the 'right' handed gestures while utterly disregarding all the 'left' handed instances captured by the webcam.

2 Related Works

Hand gesture recognition for Human Computer Interaction has been an active area of research with several approaches being proposed over the years. These approaches range from traditional image processing techniques to modern day deep-learning based systems. To draw a contrast between the two techniques we can quantify by using compute insensitivity as a metric for usage. Traditional image processing techniques are very computer intensive for each input and take a lot of time to process each image (mostly one at a time). When contrasted with the deep learning based systems, these systems are compute intensive only during the training cycle which can be improved by using batching of inputs together. Once a neural net model is trained, fitted to a desired convergence (without over-fitting), the model requires very minimal computation to reproduce result. But on the metric of data requirement, traditional methods are less data hungry when compared to a deep neural net based algorithm, as traditional models apply an algorithm directly to the input, whereas a deep neural network learns algorithm during training by generalizing and learning features from a lot of data. This section highlights two key works that I have used in this report to form the foundational baseline for comparison.

Authors of first approach[3] introduces one of the earliest gesture recognition systems by leveraging traditional image processing techniques. This model uses single camera to capture static images of various gestures which are then processed using segmentation, orientation detection and feature extraction. These features are then used to define gestures for recognition. This system was tested on 390 images and achieves an average accuracy of 92.3%. Due to traditional image processing techniques being compute intensive, each gesture detection takes 2.76 seconds on an average. Even though this foundational work demonstrates feasibility of hand gesture recognition, it was limited by its reliance on static image and slow detection time. This model will be referred to as 'Baseline-1' for comparison analysis.

The second approach [4] advances the field by incorporating machine learning algorithms into gesture recognition pipeline. Their system utilized a Kinect 3D camera to record gesture images at 20 frames per second, enabling the recognition of dynamic gestures. This approach improved the first[3] approach by adding temporal modeling that recognizes motion of gestures, on top of spatial modeling that detects one frame of a gesture. The methodology included histogram clustering for hand detection and a 3D convolutional neural net for gesture classification. While this approach improved the ability to detect gesture movement, the accuracy of the model averaged at 85.83%. Besides these decrement in accuracy, this model forms an essential basis for this project by showcasing potential of machine learning in enhancing gesture recognition systems. This model will be referred to as 'Baseline-2' for comparison analysis.

Google's Mediapipe hands[2] framework is state-of-the-art pre-trained model for real-time hand gesture recognition that leverages advance object detection algorithms. This model is built on foundational research in deep learning and compromises of two state detection model. The first stage is palm (object) detection that utilizes Single Shot Multibox Detector (SSD)[5] that localizes and classifies object in a single pass through the palm-detector network. By using FPN[6] for being able to detect objects at varying scales and minimizing focal loss[7], a large number of anchors are supported, while providing an exceptional trade off between speed and accuracy. The second stage detects 21 hand-landmarks' position in x, y and z coordinates along with handedness as well as hand presence.

MediaPipe exemplifies the advancements in object detection summarized by Kaur and Singh (2023) [8]. Their review highlights the critical role of deep learning in revolutionizing object detection through techniques like anchor-based detectors, multi-scale learning, and loss function optimizations. By integrating these advancements, MediaPipe offers real-time, accurate hand detection and tracking,

establishing itself as a benchmark framework for gesture recognition. This is the basis of considering mediapipe hands library for implementation of this project.

3 Methods

For creating a gesture controled UI/UX navigation pipeline, webcam video data is fed into the mediapipe-hands model. This model returns x, y and z coordinates of each of the 21 landmarks which is fed into a feed-forward network for classification. The gestures are classified into 7 categories namely, 'not a gesture', 'track mouse pointer', 'execute left click', 'execute right click', 'scrolling window', 'execute right click' and 'scroll switch screen'. These gestures are then piped into a mouse operation function that converts these gestures to the corresponding mouse operations, all the while tracking tip of index finger for appropriate mouse pointer synchronization. Figure 1 shows the flow for the completed Gesture recognition pipeline.

3.1 Python virtual environment

To enable the execution of the gesture recognition engine, a dedicated virtual Python environment has been set up within the mediapipe_env directory of the application code. This environment contains a pre-configured installation of all necessary libraries, modules, and dependencies required for the smooth operation of the application. By bundling these pre-installed components, the system eliminates the need for users to manually handle dependencies, which can often be a time-consuming and error-prone process.

This virtual environment was designed with portability and user convenience in mind, ensuring that the application can be deployed and run on different machines without any additional setup effort. Users can activate the environment and immediately access a fully functional system, making the gesture recognition engine more accessible to a broader audience. By leveraging this approach, the development team aimed to minimize potential compatibility issues and ensure a seamless experience for all end-users, regardless of their level of technical expertise.

3.2 Data Collection

In the training directory of the application code, a custom Python script was developed to systematically record landmark position data for hand gestures. This script captures the (x, y, z) coordinates for all 21 hand landmarks and saves the data in a comma-separated values (CSV) file format. Each recorded data point is paired with a corresponding label that identifies the specific gesture being performed. This structured approach ensures the data is well-organized and easily accessible for subsequent training, validation, and evaluation processes.

The collected data is stored within the data directory of the application code to maintain clarity and organization. A total of 4,621 data points were recorded through this process, capturing a diverse range of gestures. To prepare the data for machine learning, it was divided into three subsets: training, validation, and evaluation sets, adhering to a 70:15:15 ratio. This resulted in 3,234 data points assigned to the training set, 693 to the validation set, and another 693 to the evaluation set.

The data preparation process also included a cleaning step to ensure the quality and consistency of the dataset. Detailed information regarding the data cleaning procedures is documented in the clean_data Jupyter notebook, which provides a comprehensive guide for replicating the process if needed. Upon analysis, the testing data was found to be a representative subset of the training data, as detailed in Appendix A1, reinforcing confidence in the model's generalization capability.

3.3 GestureNet (Feed Forward Network)

A feedforward neural network, named GestureNet, was specifically designed to process gesture recognition tasks. The input layer comprises 63 neurons, each corresponding to the (x, y, z) coordinates of 21 distinct hand landmarks extracted from input data, effectively encoding spatial position information. The Rectified Linear Unit (ReLU) activation function is applied to all neurons in the input layer to introduce non-linearity, enabling the network to model complex patterns in the data.

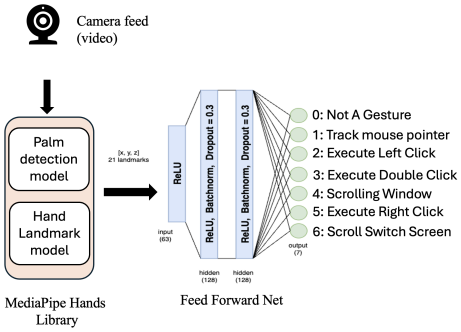


Figure 1: Application architecture.

The architecture includes two fully connected hidden layers, each containing 128 neurons. These layers also utilize the ReLU activation function, ensuring the network can learn robust features without succumbing to the vanishing gradient problem. To further enhance the training process, batch normalization is implemented after each hidden layer. This technique standardizes intermediate outputs, speeding up convergence and improving the model's stability during training. Additionally, a dropout rate of 30% is applied to the hidden layers to mitigate overfitting by randomly deactivating neurons during each training iteration. This approach ensures the network generalizes well to unseen data.

The output layer consists of 7 neurons, each representing one of the predefined gestures, with the final predictions being probabilities for each gesture class. The network was trained for 20 epochs using the cross-entropy loss function, a standard choice for multi-class classification tasks. Optimization was performed using the Adam optimizer, known for its efficiency and adaptive learning rates, allowing the model to achieve rapid and stable convergence.

After completing the training process, GestureNet demonstrated strong performance on evaluation data, achieving an accuracy of 94.08%. This high accuracy highlights the network's ability to effectively map landmark position data to gesture labels, making it a reliable component of the gesture recognition system.

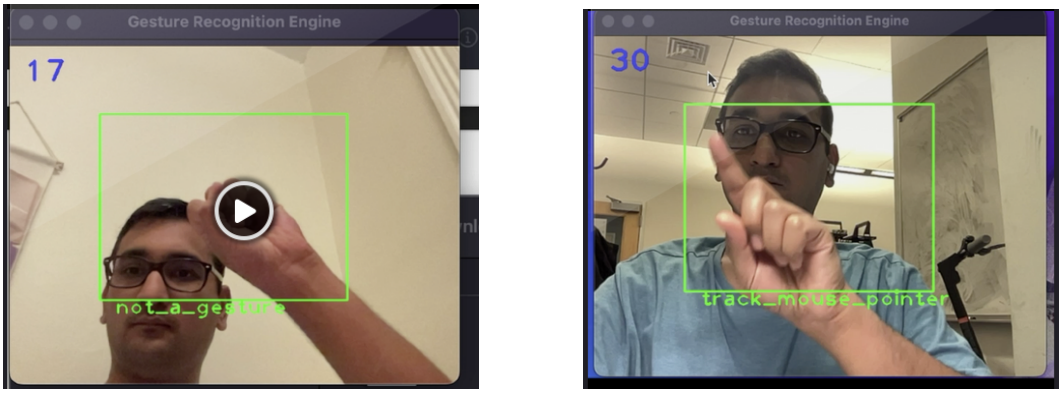
3.4 Mouse operation mapping

Mouse operations in the system were implemented using two key Python libraries: PyAutoGUI and Pynput.mouse. These libraries are well-suited for handling the low-level system calls required to simulate and manage mouse functionality programmatically. By leveraging these tools, the application is capable of executing a range of mouse operations, including pointer movement, button clicks, and scroll actions, seamlessly integrating gesture-based controls with traditional mouse functions.

To improve the user experience and enhance comfort during operation, the system includes a filtering mechanism to reduce unnecessary computations. Specifically, the camera feed is only processed by the gesture recognition model when the user's index finger is positioned within the central 50% of the camera frame, both vertically and horizontally. This ensures that only relevant gestures trigger mouse operations, minimizing false detections and preventing erratic behavior when the hand moves outside the primary area of interest.

For smoother and more precise pointer control, a tracking algorithm was implemented. The algorithm allows the mouse pointer to follow the index fingertip's movements but introduces a slight delay by reducing the tracking speed fractionally. This "smoothing" technique improves usability by granting users finer control over the pointer, particularly during delicate operations such as clicking small icons or dragging objects.

Additionally, a safeguard was added to address the issue of button bounce, a common phenomenon in which multiple button presses are erroneously registered due to rapid signal fluctuations. To counter this, the system records a single click only once every 10 frames, effectively eliminating the chances of multiple unintended clicks being registered. This implementation ensures a more reliable and frustration-free experience for users.



(a) Gesture = 0 (b) Gesture = 1.

Figure 2: Sample output from the Gesture Recognition UI/UX.

By combining advanced gesture recognition, smoothing algorithms, and robust mouse operation controls, this approach aims to deliver a user-friendly and efficient interface for gesture-driven mouse functionality.

3.5 Hypothesis on current model

On the basis of my current knowlege I am making the following Hypothesises:

H1: "This Project will take less avg time to classify one gesture". I am basing this hypothesis on the rational that the current model uses fast SSD bounding box which in itself sholuld provide a contrasting speed up when compared to Baseline-1 and Baseline-2

H2: "Landmark labels/input features of fingers used more frequently will have more importance than the rest". I am basing this on the rational that the change in value of position in landmark data will only be seen when gestures are performed. Thus fingers involved in gestures should have high importance in the output of the model.

H3: "Position coordinate ‘z’ of most of the features should not have a very high feature importance". I am basing this hypothesis on the rationale that since we are mimicking a mouse operation that happens to be a 2D operation, changes in z-position of landmarks should not be impact output a lot.

4 Experiments

4.1 Sample Output from the application

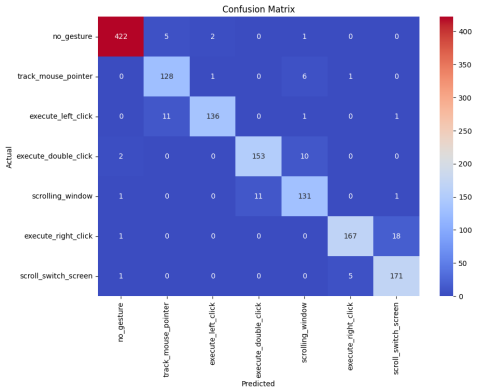
A sample output of the Gesture recognition pipeline from this project is showed in Figure 2. The green portion in the center is the gesture recognition threshold. outside this area, gesture will not be recognized. This provides two benefits: 1) mapping small movement area to mouse pointer movement on screen allows for smoother control and ease of usability, and 2) helps reduce ocular obstruction while using gesture recognition pipeline.

4.2 Empirical Analysis of GestureNet

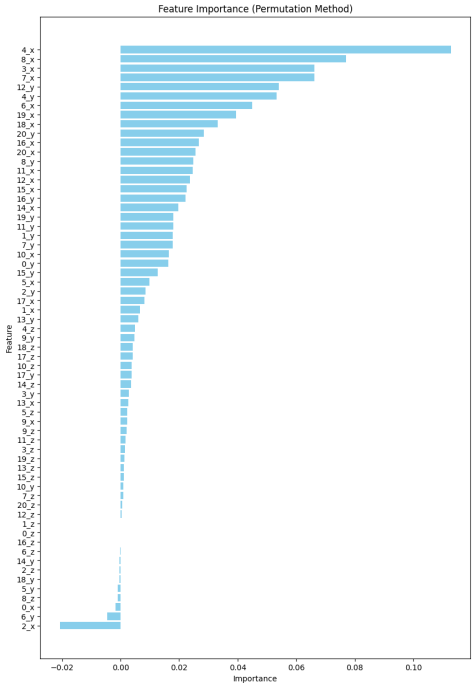
Empirical data is collected for GestureNets accuracy, The model is first compared against the 30% combined testing data and following information is retrieved. All the empirical test code is available in ‘empirical_testing’ jupyter notebook.

4.2.1 Confusion Matrix

Figure 3(a) provides a confusion matrix based on the testing data. Most predictions are concentrated along the diagonal, indicating high accuracy for correctly predicting gestures. Some notable miss-classifications between ‘execute_right_click’ and ‘scroll_switch_screen’ suggest overlap in



(a) Confusion Matrix.



(b) Feature Importance

Classification report of GestureNet model

	precision	recall	f1-score	support
0.0	0.9882903981264637	0.9813953488372092	0.9848308051341891	430.0
1.0	0.8888888888888888	0.9411764705882353	0.9142857142857143	136.0
2.0	0.9784172661870504	0.912751677852349	0.9444444444444444	149.0
3.0	0.9329268292682927	0.9272727272727272	0.9300911854103343	165.0
4.0	0.8791946308724832	0.9097222222222222	0.89419795221843	144.0
5.0	0.9653179190751445	0.8978494623655914	0.9303621169916435	186.0
6.0	0.8952879581151832	0.9661016949152542	0.9293478260869565	177.0
accuracy	0.943042537851478	0.943042537851478	0.943042537851478	0.943042537851478
macro avg	0.9326176986476439	0.933752800579084	0.9325085777959859	1387.0
weighted avg	0.9446214765352625	0.943042537851478	0.9432087781274524	1387.0

(c) Classification Report

Figure 3: Empirical analysis of GestureNet.

features between these gestures which is expected due to similarity between the two gestures. Minimal confusion exists for 'no_gesture', showing the network can distinguish non-gesture scenarios effectively

4.2.2 Feature Importance

Figure 3(b) provides a feature importance plot generated using permutation method. Features like 4_x, 8_x, and 3_x have the highest contributions, indicating that spatial information along the x-axis is critical. This also signifies that thumb and index are most relevant features for our model output. This Proves our H2.

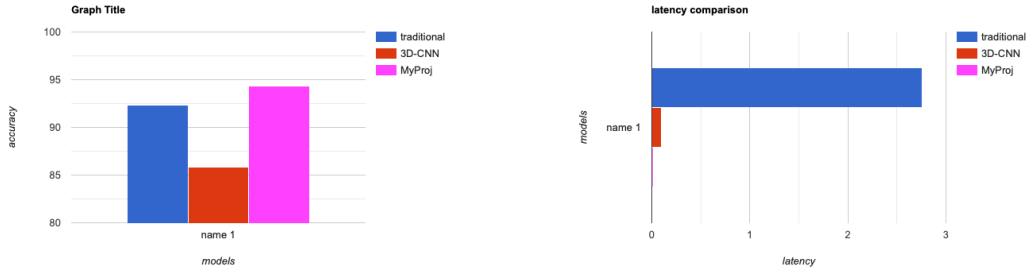
Lower importance for z-axis features suggests they contribute less to differentiating gestures, aligning with gestures being primarily planar. This Proves out H3.

4.2.3 Classification Report

The model achieves an overall accuracy of 94.3% with a macro-average F1-score of 93.3%. High precision and recall for 'no_gesture' (class 0.0) indicate reliable detection of idle states. Classes like 'track_mouse_pointer' and 'execute_double_click' show slightly lower precision as compared to recall, indicating the need for further refinement in feature extraction or training.

4.3 Comparison with Baselines

This project outperformed both Baseline-1 and Baseline-2 on accuracy and latency. This proves my H1.



(a) Accuracy graph.

(b) Latency Graph

Figure 4: Empirical analysis with Baselines.

5 Conclusion

From the results of this Project, I can conclude that mediapipe based approach turned out to be better than that of both the Baselines, because this project is using latest hardware and software libraries that have been refined over the past 10-15 years. Because the Gesture net is trained on Apple's Silicon GPU (MPU), most of the speed achieved can be associated to hardware development as well. A more detailed study required access to the program files form the two Baseline projects, which I was not able to do.

On the basis of empirical results form GestureNet, the model behaved as predicted. There is room for future improvement, especially by plugging an Adversarial Neural Net to the output of my current Gesture net and fine tune the specific parameters like 'execute_double_click' and 'scroll_window'. This will allow me to tune at granular level instead of randomly restarting the whole learning process. This is concluded on the basis of Feature Importance graph results which is in the field of Explainable AI to astute understanding of the model.

References

[1] Mistry, P., & Maes, P. (2009). SixthSense: a wearable gestural interface. In *ACM SIGGRAPH ASIA 2009 Art Gallery & Emerging Technologies: Adaptation* (pp. 85-85).

[2] Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C. L., & Grundmann, M. (2020). Mediapipe hands: On-device real-time hand tracking. *arXiv preprint arXiv:2006.10214*.

[3] Panwar, M., & Mehra, P. S. (2011, November). Hand gesture recognition for human computer interaction. In *2011 International Conference on Image Information Processing*(pp. 1-7). IEEE.

[4] Abhishek, B., Krishi, K., Meghana, M., Daaniyaal, M., & Anupama, H. S. (2020). Hand gesture recognition using machine learning algorithms. *Computer Science and Information Technologies, 1*(3), 116-120.

[5] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y. & Berg, A. C. (2016). Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14* (pp. 21-37). Springer International Publishing.

[6] Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2117-2125).

[7] Ross, T. Y., & Dollár, G. K. H. P. (2017, July). Focal loss for dense object detection. In *proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2980-2988).

[8] Kaur, R., & Singh, S. (2023). A comprehensive review of object detection with deep learning. *Digital Signal Processing, 132,* 103812.

[9] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research, 15*(1), 1929-1958.

[10] Bisht, R. (2024). *Rajatpratapbisht/gesture-controlled-UI UX*. GitHub.

`https://github.com/rajatpratapbisht/Gesture-controlled-UI_UX.git`