Vision-Based Gesture Recognition for Mouse Control

Paolo G. Estavillo, Dale Joshua R. Del Carmen, Rhandley D. Cajote
Electrical and Electronics Engineering Institute
University of the Philippines Diliman
Quezon City, Philippines
{paolo.estavillo, dale.del.carmen, rhandley.cajote}@eee.upd.edu.ph

***Abstract*—Vision-based hand gesture recognition (VGR) systems must provide the following functionalities or criteria to control a computer mouse: (i) hand tracking ability, (ii) continuous static and dynamic hand gesture recognition, and (iii) efficient resource management. Our motivation stems from the fact that only a few research so far has accommodated all these three criteria. In this paper, we developed a VGR system that accommodates these three criteria. We propose an algorithm that simultaneously detects and classifies hand gestures using RGB images and hand skeletons. To evaluate our work, we used the IPN dataset which consists of hand gestures that are suitable for mouse control. Compared to previous methods on the IPN dataset, our resulting VGR system achieves better performance in both isolated and continuous hand gesture recognition (HGR). For continuous HGR, we achieved 61.30% Levenshtein accuracy.**

***Index Terms*—hand gesture recognition, vision-based hand gesture recognition, continuous hand gesture recognition**

I. Introduction

Hand gesture recognition (HGR) is a significant area of research that covers various applications such as sign language, visual augmented reality, and medical support. With the outstanding performance of deep learning in computer vision in recent years, the use of vision-based hand gesture recognition (VGR) systems has become widespread [1]. One benefit of using VGR is that it can enable users to interact with computers through hand gestures captured by cameras. By leveraging VGR, we can enhance the way users interact with digital interfaces, especially in the context of controlling a computer mouse.

There are two basic functionalities that VGR systems must provide to control a computer mouse: (i) hand tracking, and (ii) recognizing continuous static and dynamic hand gestures. These two items are important so that users can both change the location of the mouse cursor based on their hand's location and also perform a series of natural hand gestures to execute complex actions. Most of the earlier research has been able to accommodate either (i) [2], [3] or (ii) [4], [5] but only a few works so far have accommodated both [6]. Additionally, considering that controlling the mouse cursor is expected to be a trivial task when using a computer, we also emphasize a third criterion: (iii) resource efficiency. This is so that users can use the system alongside other programs on their computers.

In this work, we have developed a VGR system that fulfills the three criteria mentioned above so that it can be used for mouse control. We propose an algorithm, based on [4], that uses lightweight deep learning models for tracking hand joint locations and predicting continuous hand gestures. Additionally, we employ MediaPipe Hands [7] to extract hand joint locations that will be later used for both tracking and predicting hand gestures.

We then evaluated our proposed algorithm on the IPN Hand dataset [5] which consists of pointing and clicking gestures, as well as other gestures that can be used to improve the flexibility of VGR systems. For our deep learning models, we used R(2+1D)-18 [8] and TD-Net [9] for gesture detection and gesture classification respectively. Compared to previous methods evaluated on the IPN dataset, we achieved the best offline classification accuracies as well as the best Levenshtein accuracy for continuous and online prediction.

The rest of the paper is organized as follows. In Section II, we discuss earlier VGR works that were used for mouse control. Then, in Section III, we discuss our proposed algorithm. Finally, Section IV discusses the results of our experiments and Section V discusses future works and concludes the paper.

II. Related Work

Earlier works in VGR used for mouse control focused on recognizing static and dynamic hand gestures. The authors in [2] performed background subtraction before feeding video frames to their 2DCNN network. The authors also used the hand centroid that was computed using a convex hull algorithm for tracking the hand location. However, the method was not able to accommodate dynamic hand gestures because only static hand gestures, such as various hand signs, were used. On the other hand, the authors in [3] also used background subtraction but did not use a deep learning approach. Instead, static hand gestures were classified based on the shape of the convex hull and the number of fingers raised.

In [4], the authors proposed an algorithm for predicting continuous static and dynamic hand gestures but did not include hand tracking. The proposed algorithm used a detector and a classifier simultaneously on a continuous stream of video frames. This is because the authors focused on classifying sequences of hand gestures. Thus, a new metric is introduced, called the Levenshtein accuracy, for evaluating predicted gesture sequences. The proposed algorithm, however, needed expensive resources for computation; it used deep learning models based on RGB frames with different modalities with large numbers of parameters.

Then, the authors of [5] proposed a benchmark dataset for continuous HGR called the IPN Hand dataset. Table I shows the 14 gesture classes with their corresponding labels and Figure 1 shows some examples of the gestures in the dataset. The authors used the method proposed in [4] to establish the state-of-the-art performance on the dataset. However, the authors used the same deep learning models proposed in [4]. Thus, the benchmark for the cost of computation became expensive.

TABLE I
GESTURES IN THE IPN DATASET

Label	Name	Label	Name
D0X	No gesture	G05	Throw left
B0A	Pointing with 1 finger	G06	Throw right
B0B	Pointing with 2 fingers	G07	Open twice
G01	Click with 1 finger	G08	2 clicks with 1 finger
G02	Click with 2 fingers	G09	2 clicks with 2 fingers
G03	Throw up	G10	Zoom in
G04	Throw down	G11	Zoom out

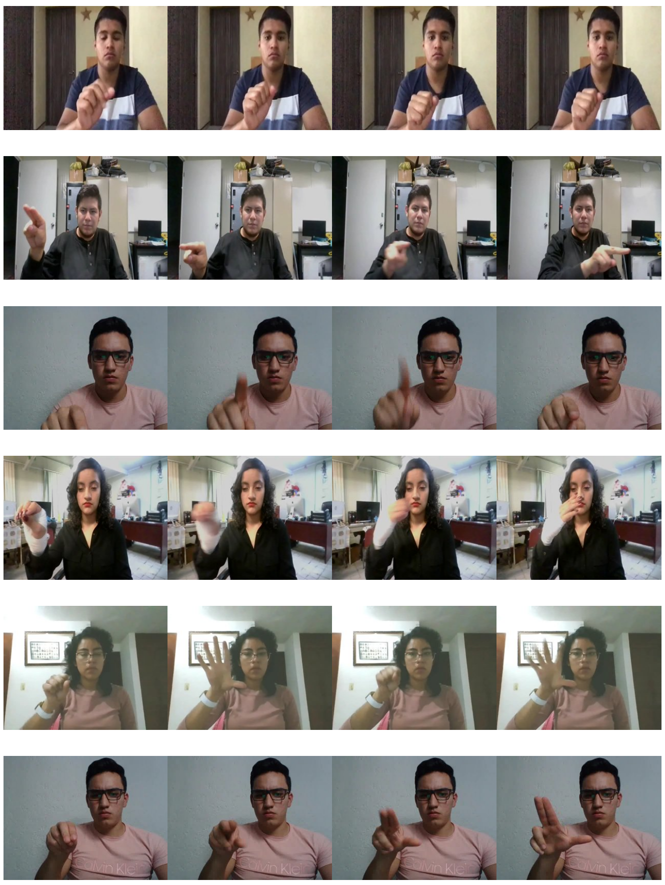


Fig. 1. Example gestures in the IPN dataset. From top to bottom: Non-gesture, Pointing with 2 fingers, Click with 1 finger, Throw left, Open Twice, Zoom in

Finally, the authors in [9] proposed TD-Net, a lightweight deep learning model that can classify static and dynamic hand gestures using hand skeletons. The model was based on DD-Net [10] but the authors added another feature, Normalized Cartesian Coordinates of Joints (NCJ), to construct TD-Net. The authors then proposed an algorithm in [6] that can accommodate hand tracking as well as static and dynamic hand gestures by using only hand skeletons extracted using MediaPipe Hands. The proposed algorithm also used a detector and a classifier; however, it met some limitations as using skeleton information only could not distinguish between gesture and non-gesture. Nonetheless, it is worth noting that the authors achieved comparable performance on the IPN dataset considering that only lightweight deep learning models were used.

In our work, we used the ideas presented in these earlier works to develop a VGR system that can be used for mouse control. To accommodate hand tracking, we used MediaPipe Hands [7] to extract hand skeletons for each video frame. On the other hand, to accommodate continuous static and dynamic hand gestures, we based our proposed algorithm on [4] by also using a detector and a classifier simultaneously. To address the expensive costs of computation, we used a skeleton classifier TD-Net [9]. Furthermore, to address the limitations faced in [6], we used a lightweight RGB detector R(2+1)D-18 [8]. Finally, we evaluated our proposed algorithm on the IPN dataset [5] for both isolated and continuous HGR tasks to confirm its robustness for mouse control.

III. Methodology

In this paper, we developed a continuous VGR algorithm that uses deep neural networks. The algorithm is based on [4] which uses a sliding window approach and two deep learning models, one as a gesture detector and the other as a gesture classifier. The gesture detector uses RGB frames only while the classifier uses hand skeletons only. The hand skeletons used by the gesture classifier are computed using MediaPipe Hands. Both the detector and the classifier are lightweight as each of them has less than one million parameters. Figure 2 shows an information flow diagram of the proposed algorithm.

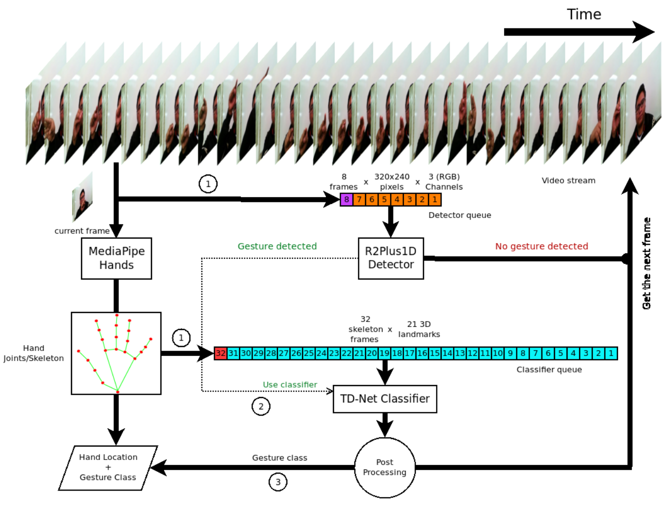


Fig. 2. Information flow diagram of the proposed algorithm based on [4]. First (1), video frames and hand skeletons are enqueued. Second (2), a gesture is detected; detector activates classifier. Third (3), classifier outputs are post-processed; gesture and hand joint locations are identified.

The algorithm starts by processing each incoming video frame one by one. First, we enqueue the video frame to the RGB detector's queue. We also compute the hand skeleton from the video frame using Mediapipe Hands and enqueue it to the skeleton classifier's queue. Second, the RGB detector processes its queue to detect whether a hand gesture is currently being performed. If it does not detect a gesture, we simply go to the next iteration and process the next incoming frame. However, if it detects a gesture, the skeleton classifier finally processes its queue of hand skeletons to identify the class of the performed gesture. At this point, we have now finished one iteration of the algorithm, so we then continue to the next incoming video frame and repeat the same process.

In comparison to other works, [4], [5] used RGB frames as input for both the detector and classifier. On the other hand, the methods in [6] used only hand skeletons for both the detector and classifier. Our algorithm acts as a compromise between these earlier methods by having the detector use RGB frames while the classifier uses hand skeletons instead.

One advantage of using hand skeletons for the classifier is that it uses less data and thus fewer parameters for classifying gestures. To classify a gesture, the classifier only uses 32 hand skeletons where each skeleton has twenty-one (21) 3-dimensional joints. In our work, our classifier only uses 517k parameters. Additionally, using hand skeletons can overcome the limitations of using RGB frames for gesture recognition, such as camera movement and varying lighting conditions. This is because, as stated in [6], skeleton information is not affected by color, light factors and environment.

It is worth noting, however, that using only skeleton information for gesture recognition also has its limitations. Skeleton information can only distinguish which gesture is performed given that the subject is performing a gesture. However, it cannot distinguish between gestures and non-gestures [6]. Thus, we built the detector such that it uses RGB frames instead to alleviate this problem.

The next sections will discuss in detail the detector, classifier, and other processes involved in the algorithm.

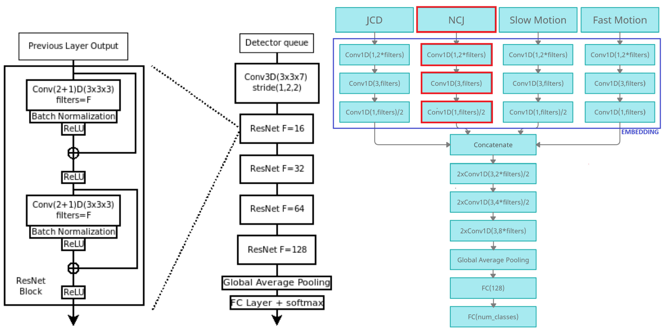
A. Sliding Window Approach

The algorithm uses a sliding window approach similar to [4] where both the detector and the classifier each use a queue of frames. Following the approach in [4], the detector queue processes the 8 most recent consecutive frames while the classifier processes the 32 most recent consecutive frames. Both queues use a stride of 1 by pushing the most recent video frame at the front and maintaining their size by popping the frame at the back.

It is also worth noting that the detector and the classifier uses different types of data as shown in Figure 2; the detector uses RGB video frames while the classifier uses hand skeletons. Thus, the queues used by the detector and classifier contains different types of data and have different dimensions.

B. Detector

The main job of the detector is to activate the classifier if it detects a gesture and to deactivate it otherwise. It is based from [8] where it uses the ResNet-18 architecture but internally uses factorized convolutions instead of full 3D convolutions. Hence the name R(2+1)D-18. This is shown in Figure 3a. With factorized convolutions, we managed to build its 18 layers using only 973k parameters. This is because considering that it continuously processes every video frame, it must be lightweight [4]. Furthermore, it uses an input sequence of only eight (8) consecutive RGB frames.



(a) R(2+1)-18 based on [8] (b) TD-Net [9]

Fig. 3. Detector and classifier models

Training Details: We first pretrain the detector with the Jester dataset [11]. Similar to the approach in [8], we trained the model with a batch size of 32 videos for 50 epochs. We used SGD as our optimizer with an initial learning rate of $1e^{-2}$ that is divided by 10 every 10 epochs. For data augmentation, we followed strategies used in [12]; for every gesture clip, we randomly selected eight (8) frames by temporally dividing the gesture clip into 8 equal parts and then randomly selecting a frame in every part. If the gesture clip is less than 8 frames, we randomly select the frames to duplicate. Additionally, for each batch we applied TubeMix with a probability of 0.5. We then trained the detector on the IPN dataset [5]. We used the same approach but since the IPN dataset has significantly fewer gesture instances, we used 100 epochs and decreased the learning rate every 20 epochs instead.

C. Using MediaPipe Hands To Compute Hand Skeletons

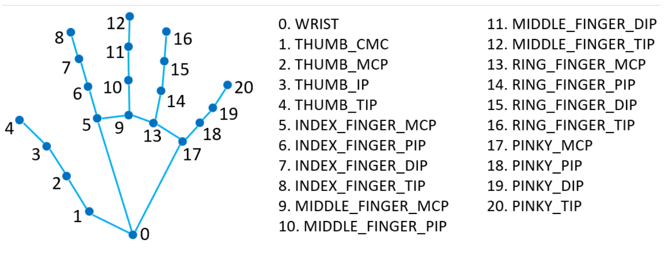


Fig. 4. 21 hand landmarks used for the hand skeleton

For computing the hand skeletons, we used MediaPipe Hands [7]. Figure 4 shows the twenty-one (21) hand landmarks that MediaPipe outputs given an RGB image. We set the detection, tracking, and presence thresholds to a low value of 15%. This is because setting them to higher values leads to MediaPipe losing track of the hand during fast hand gestures. Thus, we set the thresholds to 15% to keep track of the hand skeleton during fast hand gestures present in the IPN dataset.

Furthermore, if MediaPipe was not able to compute a hand skeleton for a certain video frame, we use a dummy hand skeleton that is filled with a value of -1. This is because MediaPipe outputs twenty-one (21) 3D hand joint coordinates in the form (x, y, z) where x and y are normalized spatial coordinates of the hand joint, and z is the estimated depth. If MediaPipe detects a hand, x and y can lie in the range $[0, 1]$

if the joint is inside the video frame. If the joint is outside the video frame, however, its x and y can be slightly below zero or slightly above one. Thus, we use a dummy skeleton filled with coordinates set to -1 to signify that no hand is inside the video frame.

D. Classifier

Once the detector activates the classifier, only then will the classifier process its queue. We used TD-Net [9] as the classifier's architecture as shown in Figure 3b. It is based from its predecessor, DD-Net [10], which consists of the Joint Collection Distance (JCD) feature, together with the Slow and Fast Motion features. To build TD-Net, the authors in [9] added the Normalized Cartesian coordinates of Joints (NCJ) feature to enrich the spatial information received by the model. The computation of JCD, and both Slow and Fast motion features are explained in [10] and the computation for NCJ is explained in [9].

When building the classifier, we used the same details proposed in [9] but we set filters equal to 32 instead of 64 and set the input sequence to 32 frames so that the network only uses 517k parameters.

Training Details: We trained the classifier from scratch with the hand skeletons extracted from the IPN dataset [5]. We followed the training details mentioned in [10]; we used Adam as our optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and trained the network for 1200 epochs. Instead of using an annealing learning rate, we used a cosine decaying learning rate initialized at $1e^{-3}$ decayed to $1e^{-5}$. For data augmentation, we used one of the strategies proposed in [12]; for each gesture clip, we randomly select 32 frames with the same approach as before and apply StackMix with a probability of 0.5.

E. Post-processing The Classifier's Output

Instead of taking the raw output of the classifier for each iteration, we followed the methods proposed in [4] for post-processing the output probabilities of the classifier. This is because, as seen in the authors' experiments, misclassifications tend to occur at the beginning of a gesture. Thus, we incorporated the methods proposed by the authors so that the gestures are classified only after their most discriminative parts are seen. The post-processing settings were set similarly to [5], with the mean gesture duration set to 65 frames for the IPN dataset.

F. Improving The Algorithm

Due to achieving low performance, we modified and attempted to improve the algorithm in Figure 2. In the modified algorithm, we double both queue lengths and process every other frame including the most recent frame instead as seen in Figure 5. This is because the authors of [4] tested their algorithm against the EgoGesture and nvGesture datasets, both having a mean gesture duration of 38.4 frames. The dynamic hand gestures in the IPN dataset, however, have a mean gesture duration of 65 frames. Thus, we double the length of the detector and classifier queues and process every other frame so that the algorithm can adapt to longer gestures but at the same time, the detector and classifier use input sequences of the same length as before.

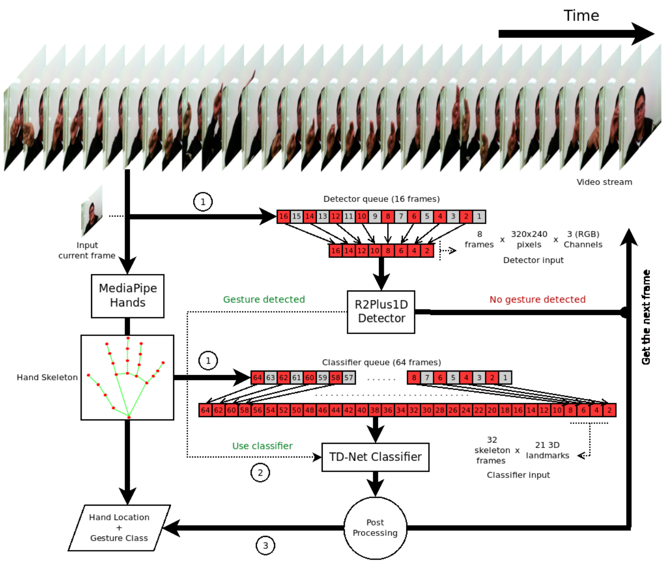


Fig. 5. Improved version of the algorithm in Figure 2. The queues are now doubled and the models now look at every other frame instead of consecutive frames.

Furthermore, during our experimentation with the IPN dataset, we noticed that not all frames are needed to fully identify a gesture. Given that our detector and classifier models only use a small number of frames, using consecutive frames does not show a significant progression in time. This is because in most cases, the next frame only has a small difference compared to the current frame. Thus, without throwing away any input frames, we modified the algorithm such that the models look at every other frame on their respective queues instead. This is so that we slightly increase the difference between two frames at the models' inputs and thus, we enrich the temporal information received by our models.

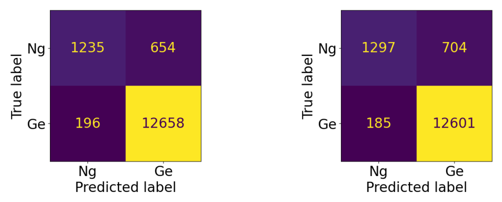
IV. Experiments

A. Isolated HGR

1) Pretraining the Detector With the Jester Dataset: The Jester dataset is a collection of 148,092 video clips where each clip depicts one person performing a single hand gesture in front of a webcam [11]. Each gesture can be one of 27 classes, 25 of which are static or dynamic, while the other two classes are not labeled as any particular movement. The other two classes are "No Gesture" and "Doing other things." The first presents a user sitting or standing still while the second is a collection of irrelevant but natural movements such as stretching, scratching the head, etc.

In our work, we labeled the 25 classes as "Gesture" (Ge) and the other two as "Non-gesture" (Ng). Additionally, we did not change how the dataset was split; we followed the default split ratio of 8:1:1 for the train, test, and validation splits respectively. As seen in Figure 6, our detector achieved similar results for the test and validation splits.

2) Results With the IPN Dataset: The IPN dataset consists of 200 videos where each video depicts one person performing multiple hand gestures one after another in front of a webcam. The dataset has a total of 5469 gestures and each gesture is labeled as one of 14 classes. 13 of these classes



(a) Test (b) Validation

Fig. 6. Confusion matrices of the gesture detector on the Jester dataset. (Left) Test split performance. (Right) Validation split performance.

are either static or dynamic hand gestures while the other one also represents irrelevant but natural movements similar to the Jester dataset. We used the same approach and labeled the 13 gesture classes as "Gesture" and the remaining class as "Non-gesture."

Table II and Figure 7 show the resulting precision, recall, and confusion matrix on the test set of the dataset. Compared to the 8-frame RGB detectors in [5], ours achieved better precision and recall with just a small increase on the number of parameters. One explanation for this is that the data augmentation techniques that we used, namely TubeMix [12], was effective for training the model. Since the frames were randomly selected and the videos were randomly mixed in the spatial dimension, the detector model was able to generalize better. Another possible explanation for this is that R(2+1)D may be better than ResNets for detecting motion. This is because factorized convolutions might model hand motions better than full 3D convolutions and that the increase in ReLU activations help increase the model's complexity.

TABLE II
COMPARISON WITH PREVIOUS DETECTOR MODELS ON THE TEST SET OF THE IPN DATASET

Model	Modality	Params.	Precision (%)		Recall (%)	
			Ng	Ge	Ng	Ge
ResNet-10 [5]	RGB	**0.895M**	60.4	76.4	14.4	96.7
ResNet-10 [5]	RGB-Flow	0.908M	87.4	81.5	36.3	98.17
ResNet-10 [5]	RGB-Seg	0.908M	73.8	80.9	35.2	95.6
R(2+1)D-18	RGB	0.973M	**95.3**	**91.5**	**80.3**	**98.18**

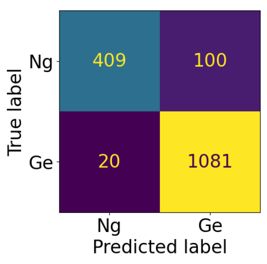


Fig. 7. Confusion matrix of our RGB detector on the IPN dataset.

On the other hand, Table III and Figure 8 show the accuracy and the confusion matrix of our classifier evaluated on the 13 classes. From Table III, we can see that we have achieved the best results, having an accuracy of 94.66% while using only 517k parameters. One explanation for this is that the data augmentation we employed, namely StackMix [12], made training more effective; by randomly selecting hand skeletons and by randomly mixing them in the temporal dimension, the classifier was able to generalize to the test data better. This is also because StackMix was not employed in training the previous models.

TABLE III
COMPARISON WITH PREVIOUS CLASSIFIER MODELS ON THE TEST SET OF THE IPN DATASET

Model	Input	Modality	Parameters	Acc(%)
ResNeXt-101 [5]	32 frames	RGB	47.51M	83.59
ResNeXt-101 [5]	32 frames	RGB-Flow	47.56M	86.32
ResNeXt-101 [5]	32 frames	RGB-Seg	47.56M	84.77
TD-Net [6]	—	Skeleton	1.88M	84.98
TD-Net	32 frames	Skeleton	**0.517M**	**94.66**

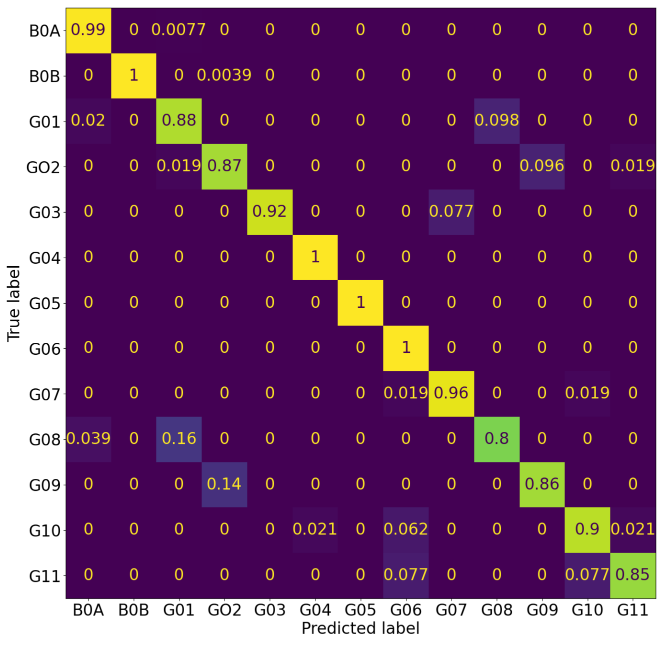


Fig. 8. Confusion matrix of our skeleton classifier on the IPN dataset.

B. Continuous HGR

For continuous HGR, we evaluated how well the proposed algorithm predicts gesture sequences. We used the Levenshtein accuracy, a metric introduced by [4] and used in the works of [5] and [6]. In brief, the metric is obtained by first computing the edit distance between the ground truth sequence and the predicted sequence. Then, by dividing the edit distance by the length of the ground truth sequence and subtracting it to one, we obtain the Levenshtein accuracy.

We evaluated the Levenshtein accuracy of the algorithm in Figure 2 and its modified version in Figure 5 with the test videos of the IPN dataset. Table IV shows the results, as well as the Levenshtein accuracies obtained in [4]–[6]. The reported values are optimized by varying the early detection threshold during post-processing. From the table, we can see that the algorithm only achieved 37.50%. This shows that better performance on isolated HGR does not result in better performance in continuous HGR.

On the other hand, the modified algorithm achieved 61.30%. This confirms that we have enriched the temporal information received by the detector and classifier models. This was because the modification has enabled the classifier to have a temporal scope of 64 frames. With 64 frames in scope, the classifier was able to fully capture the important parts of the dynamic gestures of the IPN dataset which have a mean duration of 65 frames.

TABLE IV
LEVNSHTEIN ACCURACIES OF DIFFERENT METHODS

Models	Modality	Lev. Acc(%)
ResNet10+ResNeXt-101 [5]	RGB	25.34
ResNet10+ResNeXt-101 [5]	RGB-Flow	42.47
ResNet10+ResNeXt-101 [5]	RGB-Seg	39.01
TD-Net [6]	Skeleton	40.10
R(2+1)D-18+TD-Net	RGB + Skeleton	37.50
R(2+1)D-18+TD-Net (Modified)	RGB + Skeleton	**61.30**

We also measured the number of correct predictions made in each test video to compare the recognition rate shown in Table V. We used the source code in [5] to obtain results for their ResNet-10 and ResNeXt-101. From the table, we can see that the modified algorithm correctly recognized a total of 769 out of all the 1101 gestures. We can also see that it has consistently predicted more of each gesture correctly compared to [5]. It is also worth noting that despite adjusting the algorithm only to the duration of dynamic gestures in the IPN dataset (G01-G11), it has also improved its performance on the static gestures (B0A and B0B).

TABLE V
COMPARISON OF THE RECOGNITION RATES ON THE TEST SET OF THE IPN DATASET

Label	Total Instances	ResNet-10 + ResNeXt-101 [5]			Ours (Modified)
		RGB	RGB Flow	RGB Seg	RGB + Skeleton
B0A	265	107	186	166	**212**
B0B	264	103	182	166	**220**
G01	52	7	16	16	**32**
G02	52	5	11	10	**23**
G03	52	10	25	21	**34**
G04	52	13	29	18	**33**
G05	52	10	35	29	**37**
G06	52	10	30	26	**36**
G07	52	11	21	20	**35**
G08	52	10	9	10	**25**
G09	52	4	9	10	**28**
G10	52	12	24	18	**32**
G11	52	9	18	20	**22**
Total (%)	1101	311 (28.2)	595 (54.0)	530 (48.1)	**769 (69.8)**

Overall, the results shown in Tables IV and V show the importance of considering the gesture duration for continuous HGR. Because each of the classifiers in [5] has a temporal scope of only 32 frames, they were not able to fully capture the information for significantly longer gestures such as the dynamic hand gestures in the IPN dataset which are 65 frames long on average. Consequently, the algorithm used in [5] and our straightforward implementation of it using hand skeletons in Figure 2 had difficulty in predicting gestures for continuous HGR.

V. CONCLUSION

In this paper, we introduced a VGR system based on [4] that accommodates hand location tracking, static and dynamic hand gestures, while using efficient and lightweight deep learning models for mouse control. Our experiments on the IPN dataset have confirmed that using RGB together with skeleton information is an efficient way of improving the algorithm in [4].

Our experiments on isolated HGR have shown that using factorized convolutions is a robust way for detecting hand motions. Additionally, we found out that using the data augmentation techniques proposed in VideoMix [12] were helpful for training even for skeleton classification.

Furthermore, our experiments have shown that having better performance in isolated HGR alone does not automatically result in better performance in continuous HGR. Despite the models achieving better classification performance, the algorithm significantly improved only when it was adopted to longer gestures by using frame skipping with longer queue lengths. Thus, future implementations must provide a way for dealing with varying gesture durations to bring significant improvements for continuous HGR.

Overall, the proposed algorithm still has room for improvement. Other modalities such as different color spaces, like HSV and CMYK, can be explored to improve classification. In addition to dealing with gesture durations, there are still major challenges for continuous HGR that are present, such as gesture spotting, segmentation, and extracting more meaningful hand gesture features [1]. Further research must address these challenges so that better VGR systems can be developed.

REFERENCES

[1] D. Sarma and M. Bhuyan, “Methods, Databases and Recent Advancement of Vision-Based Hand Gesture Recognition for HCI Systems: A Review,” *SN Computer Science*, vol. 2, 11 2021.

[2] M. Ranawat, M. Rajadhyaksha, N. Lakhani, and R. Shankarmani, “Hand Gesture Recognition Based Virtual Mouse Events,” in *2021 2nd International Conference for Emerging Technology (INCET)*, 2021, pp. 1–4.

[3] V. V. Reddy, T. Dhyanchand, G. V. Krishna, and S. Maheshwaram, “Virtual mouse control using colored finger tips and hand gesture recognition,” in *2020 IEEE-HYDCON*, 2020, pp. 1–5.

[4] O. Kopuklu, A. Gunduz, N. Kose, and G. Rigoll, “Real-time hand gesture detection and classification using convolutional neural networks,” in *2019 14th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2019)*, 2019, pp. 1–8.

[5] G. Benitez-Garcia, J. Olivares-Mercado, G. Sanchez-Perez, and K. Yanai, “IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition,” in *25th International Conference on Pattern Recognition, ICPR 2020, Milan, Italy, Jan 10–15, 2021.* IEEE, 2021, pp. 1–8.

[6] T.-T. Nguyen, N.-C. Nguyen, D.-K. Ngo, V.-L. Phan, M.-H. Pham, D.-A. Nguyen, M.-H. Doan, and T.-L. Le, “A continuous real-time hand gesture recognition method based on skeleton,” in *2022 11th International Conference on Control, Automation and Information Sciences (ICCAIS)*, 2022, pp. 273–278.

[7] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann, “Mediapipe hands: On-device real-time hand tracking,” 2020.

[8] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, “A closer look at spatiotemporal convolutions for action recognition,” 2018.

[9] T.-T. Nguyen, D.-T. Pham, H. Vu, and T.-L. Le, “A robust and efficient method for skeleton-based human action recognition and its application for cross-dataset evaluation,” *IET Computer Vision*, vol. 16, no. 8, pp. 709–726, 2022. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cvi2.12119

[10] F. Yang, S. Sakti, Y. Wu, and S. Nakamura, “Make skeleton-based action recognition model smaller, faster and better,” in *Proceedings of the ACM multimedia asia*, 2019, pp. 1–6.

[11] J. Materzynska, G. Berger, I. Bax, and R. Memisevic, “The Jester Dataset: A Large-Scale Video Dataset of Human Gestures,” in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 2874–2882.

[12] S. Yun, S. J. Oh, B. Heo, D. Han, and J. Kim, “Videomix: Rethinking data augmentation for video classification,” 2020.