Ensemble Training for Adversary Detection on User Keyboard and Mouse Input

Eric Detjen
University of Maryland
College Park, MD 20742
edetjen@terpmail.umd.edu

Akshaj Gaur
University of Maryland
College Park, MD 20742
agaur@terpmail.umd.edu

Aidan Melvin
University of Maryland
College Park, MD 20742
amelvin@terpmail.umd.edu

Srisrujan Penikelapati
University of Maryland
College Park, MD 20742
srujanp@umd.edu

Tony Wu
University of Maryland
College Park, MD 20742
tonywu@umd.edu

Abstract

Current device security measures primarily rely on passwords and biometric algorithms to detect foreign users, but these are insufficient once bypassed. In our paper, we present a novel method for user detection that takes 5-second sliding intervals of keyboard/mouse data and trains three different architectures: convolutional neural networks, transformers, and traditional neural networks. Our results show a strong improvement in detecting foreign users over other state of the art methods.

1 Introduction

While face detection algorithms have enhanced smartphone security, there are still considerable security challenges that face other devices, such as laptops and desktop computers. Currently, face detection algorithms can ensure that the current user is an authorized user of the device. These algorithms would deny access to any foreign actors. According to Cisco Duo's 2022 Trusted Access Report (Mascellino [2022]), about 81% of smartphones in 2022 employed a biometric feature, including facial detection. Similarly, a 2020 report by Biometrics Update (Pascu [2020]) predicted that by 2024, more than 90% of smartphones would use facial recognition technologies. While there has been considerable progress towards biometric security in smartphones in recent years, these security features have lagged in other devices, such as laptops. According to the Society for Human Resource Management (Maurer [2018]), in 2018, only 25% of users reported using biometric identification on their laptops. Even in 2024, biometric identification for laptops is often limited to premier laptops, such as Apple's MacBook.

Currently, passwords are one of the most common security features for laptops, in place of biometric algorithms, such as facial recognition or fingerprinting. However, with the rise of social engineering, simple password prompts could be vulnerable to attackers. In addition, once an attacker has bypassed the password prompt, there is a lack of infrastructure in place to identify whether the user of the device is the authorized user.

Given these limitations, a model that can correctly identify whether the user currently using the device is the intended user would be useful. The algorithm would be based on several variables, such as typing and mouse movement patterns. In practice, it would work similarly to a facial detection algorithm, where the model will be able to identify the intended user among all other users. If implemented in a practical use case, the laptop could shut down or hibernate if it identifies typing behavior that does not resemble the intended user of the device.

Preprint. Under review.

2 Related works

Observing keyboard and mouse behavior for user detection has been a key topic of study across several domains. The applications range from computer security to user identification for gaming. Within these applications, several studies have taken different approaches for the respective problems. Some studies have elected to use either keyboard or mouse data in their machine-learning models, while others have selected data in specific situations, such as hybrid scenes.

First, Da Silva Beserra et al. [2016] explored the fusion of keystroke and mouse dynamics for user identification for the online game League of Legends. They used classifiers such as k-nearest neighbors, support vector machines, multi-layer perceptron, and random forests. When combining the models, the group achieved user detection accuracies of up to 90.77%. The authors aimed to apply the proof of concept for detecting account sharing, or players playing the game on another account. While the application of the paper is different, the authors explored different results using only keyboard and mouse data.

Many other studies have also used keyboard and mouse behavior detection for user authentication and intrusion detection, which is similar to our potential applications. Antal and Egyed-Zsigmond [2018] compared action-based evaluations for intrusion detection using mouse dynamics. The study aimed to find imposters given a small dataset of mouse activity. Similar to our approach, the study extracted features from the mouse data. The raw data also followed a very similar format, as it included x and y coordinates and the mouse state. Like our study, the paper performed binary classification using neural networks. They achieved an AUC of 0.92 on the Balabit Mouse Challenge Dataset using decisions based on several actions.

Our study aims to build upon the mouse behavior detection of Antal and Egyed-Zsigmond [2018] by including keyboard data. Hashem et al. [2017] proposed a similar goal as they also hoped to pursue intruder detection through keyboard and mouse data. However, Hashem et al. [2017] used simplistic features for their classifiers, such as mouse speed and distance traveled. They also trained their model on classifiers, such as k-nearest neighbors and random forests. The researchers achieved an average AUC of 0.8761.

Finally, Shi et al. [2022] built upon the study performed by Hashem et al. [2017]. The group aimed to use keyboard and mouse behavior for user detection. But, the study was limited to scene-irrelated features on hybrid scenes. In this study, the feature extraction was unique because they augmented features extracted from a single user to scene-irrelated features. The algorithm also used a feature selection algorithm to decrease the dimensionality of the data. Similar to our proposed solution, the study employed a neural network for classification. The group achieved an 84% accuracy.

3 Data collection

To train a model on detecting computer users based on keyboard and mouse activity, we had to collect training data. Using the help of the pynput Python library, we developed keylogger and mouse tracker scripts. Over the course of one week, every group member ran these scripts continuously at all times on their computers. This resulted in two large datasets: one for all keyboard activity data and one for all mouse activity data from each user.

The mouse data collected involved three things: mouse position, mouse click, and mouse scroll events. Whenever the mouse was moved, clicked, or scrolled, the script added an entry to the log. Mouse movement logs were in the following form:

```
[datetime] move [x-coordinate] [y-coordinate]
```

An example is shown below:

```
2024-04-01 10:24:32.375540 move 546.91015625 237.5703125
```

Mouse click logs were in the following form.

```
[datetime] click [x-coordinate] [y-coordinate] [right or left] [up or down]
```

An example is shown below (showing the user clicking down on the left mouse button):

```
2024-04-01 10:24:33.133836 click 1309.5078125 473.80078125 Button.left True
```

Mouse scroll logs were in the form

```
[datetime] scroll [x-coordinate] [y-coordinate] [x-vector] [y-vector]
```

An example is shown below (showing the user scrolling downward):

```
2024-04-01 10:24:47.430066 scroll 1316.64453125 493.890625 0 -1
```

Mouse logs amounted to about 85 megabytes of data per group member over the course of the week of data collection.

Keyboard data only involved key presses, and were in the form

```
[datetime] [up or down] [x-coordinate] [y-coordinate]
```

An example is shown below (showing the user pressing the letter 'q'):

```
2024-04-01 10:38:43.844190 keyup q
```

3.1 Keyboard data preprocessing

After this data was collected, it was split into five-second intervals for feeding into the subsequent models. Only five-second intervals with at least five key presses were kept. After this preprocessing, the keyboard data was in the format of tuples of length three, where each tuple was of the form (time since beginning of interval, key pressed, up or down). The time since beginning of the interval was measured in microseconds, the key pressed was encoded to an integer, and the third value was a 1 if it was a key up event and 0 if it was a key down event.

3.2 Mouse data preprocessing

For the mouse data preprocessing, we normalized the time stamps to seconds since the first recorded time, the x and y coordinates to values between 0 and 127, the scroll speed to nonnegative integers, and the clicking actions (downward/upward, left/middle/right) to nonnegative integers. Then, we looked at sliding 5 second intervals of the mouse data every second, keeping only intervals with either at least 128 events or at least 1 click or scroll action (as these are rarer in the data). Each interval was split into 3 channels: mouse movement, scroll position/velocity, and click location/type.

For all channels, each mouse event was added to a tensor depending on its normalized location, while the intensity varies based on other factors. For mouse movement, the intensity is $55 + 40 * (t_n - t_0)$, where t_n is the time of the nth mouse event in the interval and t_0 is the time of the first event in the window. For scroll position/velocity, the intensity is $55 + \min(200, 10 \cdot v_n)$, where v_n is the velocity of the nth mouse event in the interval. For click location/type, the intensity was $55 + 24c_n + 128d_n$, where $c_n \in \{0, 1, 2\}$ is the click type and $d_n \in \{0, 1\}$ is whether the click was downwards or upwards. Figure 1 shows 3 different intervals after preprocessing, with the intervals displaying the move, click, and scroll channel respectively from left to right.

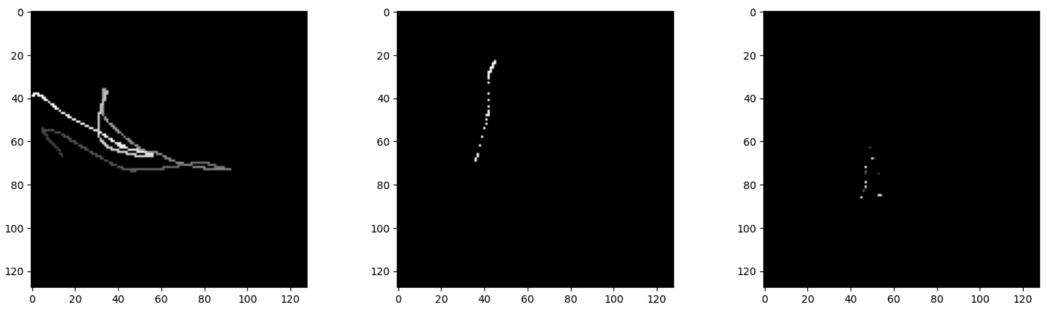

Figure 1: Move/click/scroll data after preprocessing (3 different intervals)

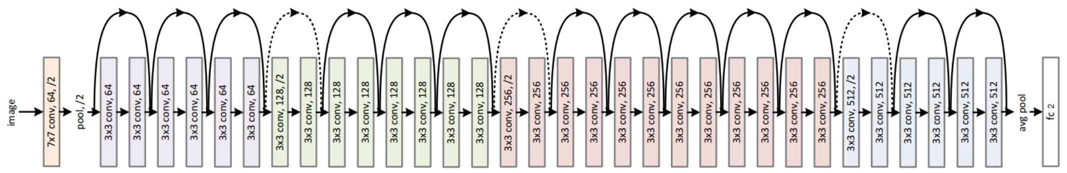

Figure 2: 34 layer residual CNN network

4 Methodology

4.1 Convolutional neural networks

For the convolutional neural network, we trained a one-versus-all (OVA) classification model. To avoid the class imbalance problem, we took 5000 intervals for label 1 (a single user) and 5000 intervals for label 0 (the rest of the users). Thus, the baseline accuracy for an untrained model is 50%. We used a train, validation, and test split ratio of 0.6, 0.2, and 0.2 respectively.

The convolutional neural network architecture (see Figure 2 for the architecture) is based off of ResNet34. He et al. [2016] showed that residual networks solve the vanishing gradient problem using skip connections between blocks, which allow gradients to flow directly backwards. The building blocks of the network are residual blocks, which consist of 2 convolutional filters, ReLU, batch norm, and a skip connection that bypasses everything, pictured in Figure 3.

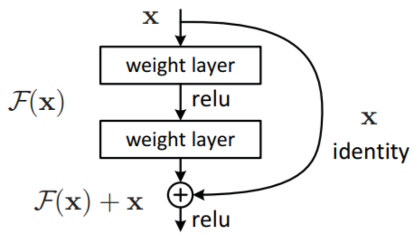

Figure 3: Residual block

The architecture uses 34 total convolutions, which are grouped into 5 layers. The first layer takes in the image and applies 64 convolutional filters with a 7×7 kernel and stride 2, ReLU, and a max pool. The next 4 layers consist of 3, 4, 6, and 3 residual blocks respectively. In each layer, all of the residual blocks take in 64, 128, 256, and 512 channels respectively. The first residual block in each layer has stride 1 and outputs the same amount of channels as the input. The rest of the residual blocks output double the amount of channels as the input, but have stride 2 to downsize the image. When the dimension increases from one layer to the next, the skip connection between residual blocks uses 1×1 convolutions, pictured as a dotted line in Figure 2. Otherwise, the skip connection is the identity function. After the last layer, we apply an average pool and then a fully connected layer with 2 outputs, which represent the two possible binary classifications.

4.2 Transformers

Second, the keyboard data was fed into a transformer model (see Figure 4 for the architecture). A transformer without a decoder was chosen because we are doing classification instead of a sequence-to-sequence or sequence generation task. Before applying a transformer to the keyboard data, the data size had to be regulated. The problem with the data size is that five-second intervals very frequently had different numbers of keys pressed, and the transformer required a fixed-length input. There were some intervals where only five or six keys were pressed, and others where dozens were pressed. To account for this, we tried two different methods: padding every interval so all intervals were equivalent in length to the maximum length interval, and truncating intervals so all intervals became the mean length. The results for each of these methods is in the Results section.

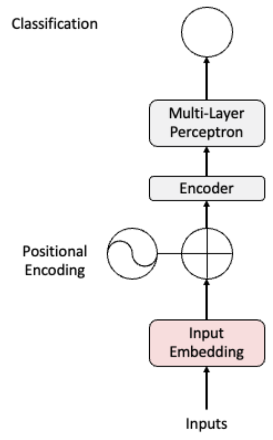

Figure 4: Transformer architecture

4.3 Neural network

Third, further feature engineering was performed on the keyboard data before it was fed into a regular, fully connected neural network for classification (see Figure 5 for the architecture). The keyboard data was chunked into five-second intervals from which features were extracted. These included the number of keys pressed, the number of non-alphanumeric keys (special keys) pressed, the average time that keys were held, the average time between depressing a key and pressing the next key, and the time between two key depresses. To better simulate a user's keyboard session, the intervals were generated using a sliding window of 1 second. This ensured our dataset would always have the most recent 5-second recording of the user's behavior. Like our other hyperparameters, we tested other sliding window sizes, and optimized on 5 seconds. The architecture of the network is as follows:

- First Hidden Layer:
  - Units: 128 neurons
  - Activation Function: ReLU (Rectified Linear Unit)
  - Dropout: 0.5
- Second Hidden Layer:
  - Units: 64 neurons
  - Activation Function: ReLU
  - Dropout: 0.3
- Third Hidden Layer:
  - Units: 32 neurons
  - Activation Function: ReLU
  - Dropout: 0.2
- Output Layer:
  - Units: 32 neurons
  - Activation Function: Sigmoid

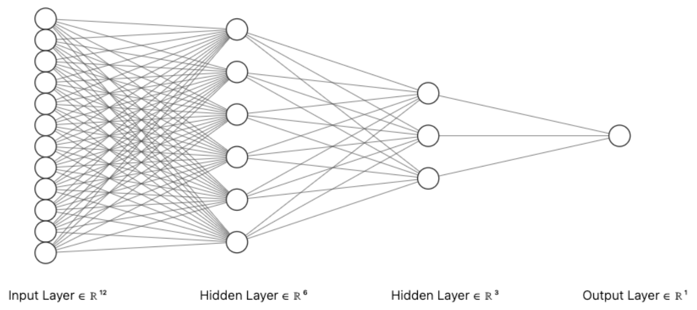
Input Layer $\in \mathbb{R}^{12}$
Hidden Layer $\in \mathbb{R}^{6}$
Hidden Layer $\in \mathbb{R}^{3}$
Output Layer $\in \mathbb{R}^{1}$

Figure 5: Neural net architecture (1:10 scale)

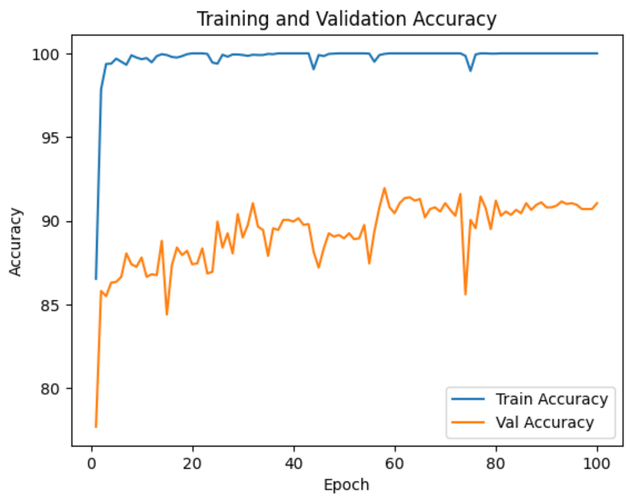

Figure 6: CNN training and validation accuracy

5 Results

5.1 Convolutional neural network results

The convolutional neural network model on the mouse data achieved an accuracy of 91.3% on the test dataset. We used a cross entropy loss function and used the Adam algorithm for stochastic optimization with a learning rate of 0.00001. We trained the model for 100 epochs with a batch size of 32. The resulting training and validation accuracy is shown in Figure 6.

Table 1 shows a confusion matrix for performance of the model on various interval sizes. We define a positive label as detecting a foreign user and a negative label as detecting the authorized user. We see that given just 1 interval, the model has a 0.960 precision and 0.862 recall, meaning that there are relatively more false negatives than false positives. This is good because we want our model to be cautious as users do not want to be falsely locked out of their own device. We also tested the performance of the classifier depending on the number of intervals given. In this case, increasing the number of intervals means separately classifying each interval and them determining the final predicted label through majority vote.

As the number of intervals increases, we see all metrics (accuracy, precision, recall, f1-score) increase. At 27 intervals, we get a 98.6% accuracy, with a 100% precision and a 97.3% recall. This shows that having just 27 5-second intervals with sufficient amounts of data allows the model to detect foreign users with extremely high accuracy. For practical use cases, this provides good evidence that the number of intervals can be increased arbitrarily to increase the accuracy.

Table 1: CNN confusion matrix

Intervals	Predicted	Actual		Accuracy	Precision	Recall	F1-score
		Positive	Negative				
1 interval	Positive	862	36	0.913	0.960	0.862	0.908
	Negative	138	964				
3 intervals	Positive	292	7	0.928	0.977	0.877	0.924
	Negative	41	326				
9 intervals	Positive	103	2	0.955	0.981	0.928	0.954
	Negative	8	109				
27 intervals	Positive	36	0	0.986	1.000	0.973	0.986
	Negative	1	37				

5.2 Transformer results

The transformer model on the keyboard data achieved a maximum validation accuracy of 35.7%. We tried transformers both with and without positional encoding, and both with sequence padding and truncation, and could not improve accuracy over this level (see Figure 7). Even though this accuracy is not very high, it is still an improvement over a 25 percent base guess accuracy (the model was trained to predict individual user rather than one-versus-all classification), which shows that the models were able to learn some features. However, on its own, this transformer could not be used in a real-world user detection system.

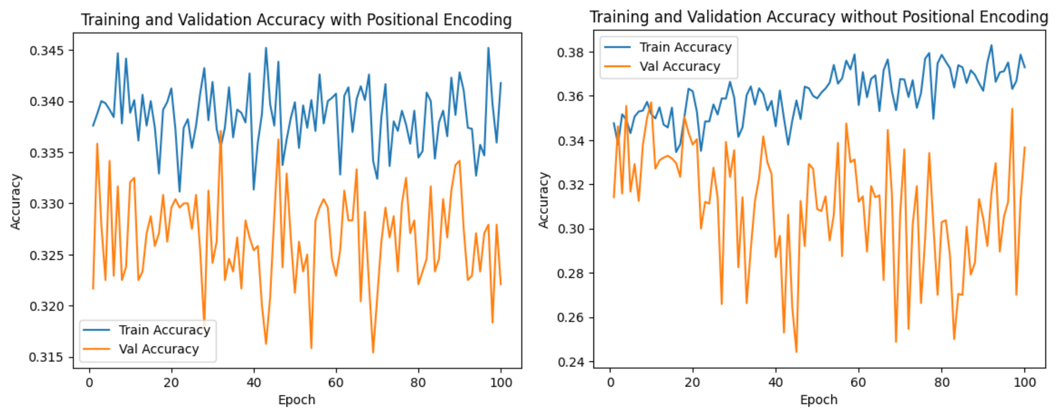

Figure 7: Transformer Training and Validation Accuracy

5.3 Neural network results

The neural network model on the keyboard data achieved a maximum validation accuracy of 30%. We tried adding additional complexities in order to further separate the data such as combining the mouse and keyboard data, though that ultimately caused overfitting. The mouse and keyboard data was also very large in size, making it quite difficult to feature engineer and train. We ultimately had to purchase a memory-rich virtual machine to avoid the standard Collab runtime crashing. With more time and compute, the combined dataset could yield significant results. Nonetheless, the maximum validation accuracy of 30% for our keyboard-only data set, while low, is still above our benchmark of 25%. Figure 8 shows the training and validation accuracy over 200 epochs.

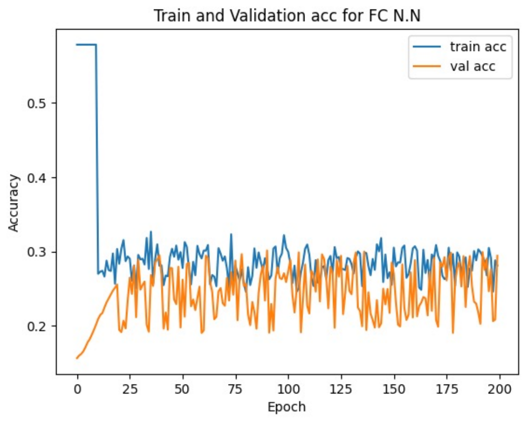

Figure 8: NN training and validation accuracy

6 Conclusion

The three methods we employed had quite different results. The convolutional neural network on the mouse data was by far the most successful, achieving a test accuracy of 91.3% over a baseline of 50% (one-versus-all classification) given just one 5-second interval. The transformer model achieved a peak validation accuracy of 33%. Finally, the regular neural network model with engineered features as input achieved a peak validation accuracy of 30%.

Although the performance of the CNN is quite impressive, the accuracy metrics might be deceptive. Even after normalizing for screen sizes (because each group member's screen had different resolutions), there are other factors we believe could allow the model to shortcut and identify user based on other things than the features we desired (mouse speed, typing speed, typing timing, clicking patterns). Each group member has different computer setups, such as different numbers of monitors and different operating systems. For example, Mac users would likely have their mouse in the top-left of the screen more often, as that is where window functions (minimize, close, maximize) are located, while Windows users would likely have their mouse in the top-right of the screen more often for the same reason. Another difference is the keyboard shortcuts that each operating system has. In the future, to control for these issues, we would collect data by having everybody use the same computer for a period of time.

7 Broader impacts

This paper presents work whose goal is to improve the security of laptop and desktop computers against foreign unauthorized users. This work specifically improves security even after logging in, providing a continuous defense against harmful adversaries.

References

[1] Alex Mascellino. Cisco report: 81 percent of all smartphones have biometrics enabled. *Biometric Update*, 11 2022. URL `https://www.biometricupdate.com/202211/cisco-report-81-percent-of-all-smartphones-have-biometrics-enabled`.

[2] Luana Pascu. Biometric facial recognition hardware present in 90% of smartphones by 2024. *Biometric Update | Biometrics News, Companies and Explainers*, 1 2020. URL `https://www.biometricupdate.com/202001/biometric-facial-recognition-hardware-present-in-90-of-smartphones-by-2024`.

[3] Roy Maurer. More employers are using biometric authentication. *SHRM*, 4 2018. URL `https://www.shrm.org/topics-tools/news/technology/employers-using-biometric-authentication`.

[4] Isaac Da Silva Beserra, Lucas Camara, and Marjory Da Costa-Abreu. Using keystroke and mouse dynamics for user identification in the online collaborative game league of legends. In *7th International Conference on Imaging for Crime Detection and Prevention (ICDP 2016)*, pages 1–6, 2016. doi: 10.1049/ic.2016.0076.

[5] Margit Antal and Elöd Egyed-Zsigmond. Intrusion detection using mouse dynamics. *IET Biom.*, 8:285–294, 2018. URL `https://api.semanticscholar.org/CorpusID:52961598`.

[6] Yassir Hashem, Hassan Takabi, and Ram Dantu. Insider threat detection based on users ' mouse movements and keystrokes behavior. 2017. URL `https://api.semanticscholar.org/CorpusID:91172836`.

[7] Yutong Shi, Xiujuan Wang, Kangfeng Zheng, and Siwei Cao. User authentication method based on keystroke dynamics and mouse dynamics using hda. *Multimedia Systems*, 29:1–16, 10 2022. doi: 10.1007/s00530-022-00997-5.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.