Spring 2023

# Enhancing Deep Learning Classifiers for Dynamic Keystroke Authentication via GANs

Jonathan A. Bazan
*San Jose State Uni*

.

# Enhancing Deep Learning Classifiers for Dynamic Keystroke Authentication via GANs

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Jonathan A. Bazan

May 2023

The Designated Project Committee Approves the Project Titled

Enhancing Deep Learning Classifiers for Dynamic Keystroke Authentication via GANs

by

Jonathan A. Bazan

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2023

| | |
|---|---|
| Dr. Katerina Potika | Department of Computer Science |
| Dr. Mark Stamp | Department of Computer Science |
| Dr. Thomas Austin | Department of Computer Science |

## ABSTRACT

Enhancing Deep Learning Classifiers for Dynamic Keystroke Authentication via GANs

by Jonathan A. Bazan

Leveraging machine learning for biometric authentication is an area of research that has seen a lot of progress within the past decade. Keystroke authentication based on machine and deep learning classifiers aims to develop a robust model that can distinguish a user from an adversary based on typing metrics (keystrokes). While keystroke authentication started with static text, where people type the same data, the shift has been to dynamic data where every user's data varies. Recent literature has shown that with enough data, deep learning classifiers have the capacity to authenticate users with a low Equal Error Rate (EER).

However, popular deep learning classifiers are bottlenecked by the large amounts of data needed to make it efficient. This work solves the problem of the data bottleneck by utilizing Generative Adversarial Networks (GANs) to generate keystroke data with a valid label. Furthermore, the synthetic data produced by the GANS are used to train the Convolutional Neural Networks (CNN), attempting to push the EER rate even lower and resolve the data bottleneck.

## ACKNOWLEDGMENTS

I would like to extends thanks to my committee members Dr. Potika, Dr. Stamp, Dr. Austin. To my former professor Dr. Aliasgari and former research colleague Joshua Kasanjian. To my family and friends who made this possible.

## TABLE OF CONTENTS

**CHAPTER**

## LIST OF TABLES

## LIST OF FIGURES

# CHAPTER 1

## Introduction

As society's reliance on technology is higher than ever before, nefarious actors are frequently leaking and/or cracking passwords at increasing rates, enabling others to gain unauthorized access and possibly cause irreparable damage to victims. Even with additional layers of security such as 2 Factor Authentication (2FA), simply stealing a smartphone can allow you to bypass this mechanism. Consequently, to enhance current security protocols, vast research in keystroke authentication has been undertaken. Keystroke authentication is a form of biometric authentication, where an individual's keystroke data, while typing, is collected by a computer and fed into a machine learning model. The models capture specific details about keystroke patterns to discriminate against unauthorized users. Moreover, as a user types their password repeatedly, the metrics used to gather information about the patterns become more stable, eventually reaching a point with minimal change in either direction. As a result, if an adversary can figure out a user's password, it is likely that their typing habits will be distinct enough for a model to recognize, especially if it's the adversary's first time typing the password. Essentially, with vast amounts of data, a computer can recognize how a group of individual's type and distinguish them from one another, even if the individuals are typing different things.

A significant amount of recent work has been centered around exploring techniques to extract features that are more representative of a user's typing habits without losing accuracy or increasing computational cost. Most of the work carried out analyzes either static or dynamic behavior. The key distinction between static and dynamic behavior are in the datasets. Static behavior is analyzed from a collection of individuals typing the same phrase/password where the typing behaviors are virtually identical. Since everyone types the same phrase, translating these metrics into a fixed-feature

vector is straightforward. In contrast, dynamic behavior is extracted from dynamic data which usually consists of a collection of users who typed very different things. Consequently, working with dynamic data means that more thought must go into the feature engineering, as variability in keystroke length as well as the keystrokes themselves may impact the effectiveness of any model. Overall, both suffer the issue of requiring a large amount of data, generally more than a user would be willing to give up time for, and a relatively high false negative rate making it impractical to apply as an extra security layer in a commercial environment.

This project builds upon recent robust deep learning architectures such as CNNs with varying kernel sizes and cutout regularization that leverage novel feature engineering techniques for dynamic keystroke authentication presented in [1]. We aim to enhance the accuracy of these classifiers to investigate if the data bottleneck issue can be mitigated through generated images by experimenting with the different Generative Adversarial Networks (GAN) architectures of the Deep Convolution GAN (DCGAN), Wassertertein GAN (WGAN), and Conditional GAN(CGAN). Additionally, this paper uses novel feature engineering techniques to transform free-text data into fixed feature vectors that can be trained on popular classical methods for static keystroke authentication like Random Forest (RF), Support Vector Machines (SVM) and K-Nearest Neighbors. We then contrast the performance of deep learning and classical methods with previous literature, by reviewing the performance, and applicability in a commercial environment. The contribution of this paper include the following:

- A novel feature engineering technique for transforming dynamic keystroke data into fixed feature vectors
- Generating dynamic keystroke data for deep learning classifier with various GAN architectures
- Using augmented data to enhance the performance of deep learning classifiers

The rest of the paper is structured as follows: Chapter 1 discusses what keystroke authentication is and the scope of the paper. Chapter 2 provides more depth into the the various forms of keystroke authentication, the data collected, and the learning techniques employed in the paper. Chapter 3 focuses on different feature engineering techniques used to transform the data for training and testing. Chapter 4 highlights the architectures hyperparameters that were selected for classification models and generative models. Chapter 5 provides an analysis of the results collected from all the experiments conducted. Lastly, Chapter 6 highlights the main goals achieved in the paper and directions for future work.

# CHAPTER 2

# Background

## 2.1 Related Work

Research in keystroke dynamics has been present for a long time, first explored in 1977 to investigate whether users could be distinguished based on the way they typed their name by [2]. However, only recently has the field of keystroke-authentication seen meaningful progress. The work of [3] has played a pivotal role in this field by enabling succeeding research to be directly comparable by establishing a static-text dataset that would serve as a benchmark. This data consisted of 50 users typing the same password - ".tie5Roanl" - 400 times and leveraging various architectures for their top performing detectors were K-Nearest Neighbors (KNN) and Support Vector Machines (SVM) to achieve an Equal Error Rate of 6-7%. Other work such as [4] sought to take keystroke authentication beyond the realm of the desktop/laptop keyboard and explore its application on mobile devices. With more data gathered through internal sensors such as fingers positioning, length, width, the models trained achieved accuracy varying from 58-91% for each user.

Moreover, studies such as [5] [6] went beyond the realm of classical methods, focusing on the utility in deep learning methods for keystroke-authentication. Consequently, deep-learning centric literature used on fixed-data like those collected in [3] revealed that these types of classifiers could outperform classical methods. [5] experimented with deep learning methods achieving an overall accuracy of 92.6%, outperforming classical methods by a considerable amount. Similarly, [6] experimented with novel feature engineering techniques that entailed transforming fixed feature vectors into multiple channeled digraphs - treated as images - into different classifiers achieving accuracy anywhere from 90-95% defeating benchmark models.

Recently, the direction in this field has moved towards creating models that can

learn and discriminate a person's typing habits via free-text datasets. Consequently, these solutions, if robust, are more practical and effective in a commercial environment. Furthermore, since datasets will contain significant variation between the samples of each user, feature engineering becomes more involved as free-text must be extracted, categorized, normalized and transformed into fixed feature vectors for classical methods. [7] uses a novel way of categorizing a sequence of keystrokes per user and majority-vote technique for classical machine learning (ML) methods, achieving a perfect accuracy for every user. However, these results are not comparable as their dataset and keystroke extraction techniques are not available. Alternatively, [8] [1] take a different approach by focusing on different keystroke features, lengths, and deep learning architectures to improve model performance.

While both do achieve great results, the feature engineering technique and the classifiers constructed in [1] achieve a considerably low EER rate that may be enhanced to meet the the European standard for access-control systems (EN-5013301), that states a detector must perform with a false negative rate of less than 1 with a false positive rate of at most 0.001. Moreover, these classifiers require a vast amount of keystroke data in order to meet the current standard, which is commercially infeasible. This paper aims to explore several GAN architectures [9] [10] [11] in conjunction with these deep learning architectures to see if the classifiers can be enhanced more, and if the data generated can be leveraged for reducing the amount of user samples required for training and testing (reducing data bottleneck).

### 2.2 Dataset

This paper consists of two open-source free-text datasets provided by SUNY university, and will be discussed further in this section.

### 2.2.1 Buffalo Keystroke Dataset

This dataset was gathered by the researchers at the University of Buffalo, consisting of 148 subjects. The subjects participated in three different laboratory sessions, spanning over a month, with 73 people using the same keyboard and 75 people using different keyboards each session to complete two tasks. One task was completing Steve Job's commencement speech broken up into three parts, and the other was completing free-text questions. The results were long text files recording the character activated, if it's pressed/released (KeyUp, KeyDown), and the timestamp of when it was pressed. Overall, about 5,700 keystrokes are gathered per subject consisting of static and dynamic responses.

## 2.3 Deep Learning Techniques

This section briefly highlights the deep learning techniques employed in the paper and their applicability for the overarching goal.

### 2.3.1 CNN

A convolutional neural network (CNN) is a deep learning model that is aimed at mimicking the human visual cortex and the way it processes information. Consisting of multiple layers including convolutional, pooling, and fully connected layers, the architecture enables the identification of spatial patterns and structures within images. Subsequently, the ability to learn and extract useful information from these images allows these classifiers to perform complex image classification tasks such image classification, object detection, and even natural language processing. The capabilities of a CNN are inherently due to its convolutional layers which allow for the classifier in training to extract important aspects of the image (textures, lines, edges) through the use of mathematical operations between a series of kernels and the input data. Moreover [1] successfully applied CNNs to dynamic keystroke authentication achieving

considerable accuracy with a low Equal-Error-Rate. This paper utilized the same CNNs for testing the quality and efficacy of the generated data.

### 2.3.2 GAN

Generative Adversarial Networks are a recent advancement in deep learning that has been a powerful tool for quality image generation across diverse and complex datasets, is a Generative Adversarial Networks (GAN). A GAN contains a unique architecture, consisting of a generator used to generate images and a discriminator to distinguish between the real and generated images. Together, these two networks go back and forth in a min-max game where the generator starts out with a random noise vector that eventually transforms into the shape of the desired input, and the discriminator evaluates the real samples against the generated and providing feedback that the generator uses to enhance the quality of the images. While the original GAN paper [12] employed Multilayer Perceptrons (MLP) in their networks and did obtain images representative of those in the MNIST dataset. The following years have produced new architectures such as a DCGAN, WGAN, and CGAN, all aimed at mitigating limitations of its predecessors. By employing different GANs in dynamic keystroke authentication, their ability to learn complex patterns and variations in data to accurately can be leveraged to reproduce keystroke data representative of a given user.

### 2.3.3 DCGAN

Deep Convolutional Generated Adversarial Network (DCGAN) is another deep learning model that has influence from CNNs using convolutional layers in the discriminator allowing it to extract features from images the same way a CNN does. Conversely, the generator uses deconvolutional layers (also known as upsampling) to take a low-resolution image and output a higher quality image that could fool

the discriminator. The deconvolutional layers work opposite of convolutional layers, performing mathematical operations between the input data and kernels, but producing an output that is greater than the input. As seen in [9], realistic 3-D images that take account of textures, shapes, and colors were produced from their GAN highlighting its robustness to learn from diverse datasets. With a powerful underlying structure, DCGANs are a popular choice for complex tasks such as high-quality image generation, or in this case keystroke generation as GANs like this allow for multidimensional input.

### 2.3.4 WGAN

Wasserstein Generative Adversarial Network (WGAN) is another GAN that was implemented with the intention of improving the overall stability and convergence during training - obstacles of previous architectures. WGAN achieves this by using a different loss function known as the "wasserstein" loss rather than the common binary cross-entropy loss. The Wasserstein distance is a powerful function as it measures the difference between two probability distributions, computing the minimum amount of work to transform one distribution into another like generated low-resolution images to high-quality samples. Moreover, as the model has the discriminator and generator alternate to update for a fixed number of iterations, this process enables the WGAN to avoid instability, the vanishing gradient, and mode collapse. [10] demonstrates these qualities by producing images on par with those generated from a DCGAN (better in some instances) but with improved stability and convergence. As a result, WGANs are a common approach when other architectures like DCGAN or CGAN succumb to the common pitfalls during training.

### 2.3.5 CGAN

Another recent and popular GAN architecture is the Conditional Generative Adversarial Network (CGAN), developed with the intention of overcoming the previous limitations such as generating data from a random noise input vector. Moreover, this limitation is overcome with the introduction of conditional input, which is just additional input such as a label or attribute that is concatenated with the random noise vector. By leveraging these conditional inputs, generating images with labels such 'happy' or 'animal' are not completely random as they will follow a pattern based on the additional input, thus eliminating the lack of ability over the content being generated in [12]. As seen in [11], the GAN conditioned with class labels was able to produce visually hyper-realistic images similar to those in the MNIST dataset, capturing a higher degree of coherence and specificity lacking in the original GAN. For dynamic keystroke authentication, the CGAN's ability to incorporate conditional input could be particularly advantageous for generating keystroke patterns that are specific to individual users typing habits.

## 2.4 Classical Learning Techniques

For this research, a limited scope of classical learning techniques was leveraged in conjunction with bagging and boosting. We briefly go over these techniques in this section.

### 2.4.1 Logistic Regression

Logistic Regression is a popular supervised learning technique for binary classification that uses a statistical model to output the probability a sample X belong to class Y. In contrast to linear regression which has a continuous output, the output of the linear relationships in logistic regression are fed into a sigmoid function to covert the output in a probability between 0 and 1. Overall, logistic regression are a good

starting point for binary classification problems, but do suffer under the assumption linear relationships exist between the features and label.

### 2.4.2 K Nearest Neighbors

KNNs are another form of supervised learning algorithm that uses a neighborhood classification like other methods like decision trees. At its core, a KNN utilizes labeled data and distance algorithm (Euclidean) to classify a set of X points based on the K elements closest to X. Given a new input data point, the algorithm searches the training data for the K data points that are closest to the input data, and assigns the class based on those neighbors. Overall, KNN is a simple and intuitive method to implement but struggle for the computational overhead as data increases and more distances must be calculated.

### 2.4.3 Support Vector Machines

SVMs are powerful supervised learning algorithm are built on the fundamentals on linear algebra and can be leveraged for tasks such as classification. In contrast, to other techniques, the internal structure relies on using kernel functions to transform the data into a higher feature space that may be linearly separable or close. Moreover, this algorithm finds a hyperplane through margin maximization function, determining the greatest margin - the minimum distance between the two classes - with the data points plotted in the feature new space. Overall, SVM is a robust algorithm that can be essential when dealing with data that tends to overfit.

### 2.4.4 Random Forest

RF are another popular form of supervised learning that build build on the structure of decision trees to produce a model with better generalization and less overfitting. This algorithm works by using an ensemble approach (Bagging) where, during training, multiple decision trees are constructed based on a random subset of

the data and features. Moreover, this inherit randomness helps the model to generalize the data better as the output of many decision trees are aggregated to achieve a classification. Overall, is a powerful algorithm to use when dealing with noisy data or non-linear relationships, but comes at a computational cost.

# CHAPTER 3

## Data Preprocessing

As touched upon in the previous chapter, the two types of keystroke datasets used for exploring ML-based keystroke authentication are static and dynamic. Each involves different approaches for pre-processing the data, especially if both deep learning and traditional techniques are employed. Since data gathered for this paper consists of free-text data, this chapter will go more into depth of the process of feature engineering.

### 3.1 Static and Dynamic Data

Although the paper uses dynamic data for keystroke authentication, it is worth briefly mentioning how different the structure of static data is compared to dynamic data, to observe why feature engineering is more involved. As the name suggests, static data consists of metrics (features) extracted from users typing the same things. In the case of [3], the participants had to type the same password ".tie5Roanl" 400 times, where each attempt yields a feature vector containing the different timing metrics extracted (31 features). As a result, the data pre-processing aspect is less intensive as the input data collected from every user yield fixed-feature vectors. Conversely, dynamic data is usually comprised of a long series of various keystrokes pressed (either up or down) with a timestamp assigned next to them, as seen in the Buffalo Keystroke dataset. Moreover, the words that people typed and the total length of keystrokes used will vary, as there are significant underlying patterns in their typing habits. For this reason, extra data pre-processing steps must be performed to transform the dynamic data into fixed feature vectors for traditional models and tensors for deep learning models like a CNN.

### 3.2 Time Based and Touch Based Features

Similar to how keystroke authentication is separated into either static or dynamic data, the type of features in most studies are also split between time and touch based features. As the name suggests, time based features involve extracting features produced from time based measurements. If a user is typing a password, the timing of the press / release between two consecutive keys, "a" and "b", would be the time based features for this pair of characters. Conversely, touch-based features are measurements taken from an individual physically interacting with a touch-screen. In the case of the same user typing their password, the touch-based features could be the pressure of tapping a letter and how much of the screen is taken up by pressing a letter. Given that most research is conducted with time-based features, studies like [13] [14] [15] [16] have shown that adding touch-based data can enhance the detection system on a smartphone.

However, there are various use-cases where touch based features selected could have a negative impact on the model being trained. For example, the same user typing the password on a hot and cold day, could produce dramatically different results that will only confuse the model in the training. Similarly, other conditions like lack of sleep, injury, sharing the same device, are all common use cases where taking touch or time based measurements can lead to a poorly performing model. In the case of time based measurements, even something simple as using a new keyboard, new keys, or a new layout, can heavily influence the results produced. As the dynamic data from the Buffalo dataset only consists of typed characters with a timestamp attached to them, feature engineering with time based features will be explored.

### 3.3 Dynamic Data Time Based Features

Similar to [7], time-based features will be extracted from each pair of consecutive keystrokes. Each feature refers to some duration of an action (press or release) between these two characters. The figure below depicts an example of typing two characters, releasing a key will produce an "Up" event whereas pressing a key yields a "Down" event. Altogether, the "Up" and "Down" events between the characters "A" and "B" will produce 5 features that fall into one of the following categories.

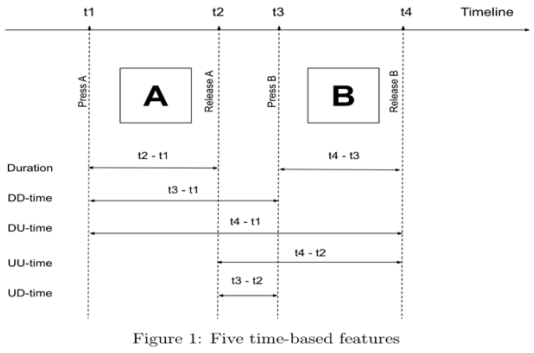

Figure 1: Five time-based features

Figure 1: Time Based Feature Extracted

- Up-Down - The time between the first key being released and the second key being pressed
- Up-Up - The time between the first and second key being released
- Down-Up - The time between the first key being pressed and the second key being released
- Down-Down- The time between the first and second key being pressed
- Duration - The time between a single key being pressed and released

### 3.4 Feature Engineering for Deep Learning Models

As mentioned in the prior chapter, this paper has significant influence from [7] and will be using the same data preprocessing technique for transforming a sequence of keystrokes collected into an image-like (5-D) tensor, called a Dynamic Keystroke

Image (DKI) that can be used as input for a CNN. Moreover, the process of this transformation will be discussed more thoroughly in this section.

### 3.4.1 Dynamic Keystroke Image

The novel feature engineering employed by [1] starts with taking a subsequence of keystrokes from the total keystrokes types by a given user, and translating those keystrokes being typed into a digraph. The most integral part of DKI revolves around creating digraphs that will be used as an input channel for the DKI. Each digraph consists of the top 42 most used characters on a keyboard and represents the duration of an event happening between two keys typed. For example, typing "A" and "B" would yield a digraph where the "AB" / "BA" position has been updated. With this example in mind, each digraph will represent one of the 5 different time features that will be extracted, resulting in 5 channels (matrices) or a 5-D image. Following the previous example, typing those two characters would update the same position in every digraph, but with different values. A more concise illustration is provided below depicting what a plain DKI looks like. Since the average amount of keystrokes per person in the dataset is 5,000+, by taking subsequences of keystrokes within the interval [50,75,100], a reasonable amount of data can be captured and updated in the DKI without adding too much noise. In the case of 50 keystrokes, a sequence of 49 could produce as many as 294 features. However, due to repeated character pairs and other factors, the number of features is always less.

## 3.5 Feature Engineering for Classical Models

As traditional models typically work with fixed feature vectors, working with a set of user DKIs would not be possible unless a way for transforming the DKI into a fixed-feature vector was found. Moreover, fixed-feature vectors were produced from the images, and were able to be trained and tested on robust classical models. This

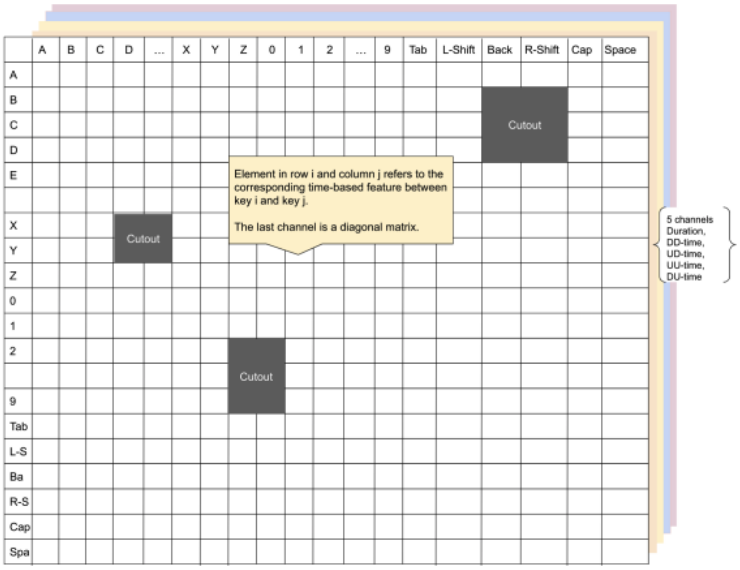


Figure 2: Dynamic Keystroke Image

section will further discuss how the key-pair mappings in the DKI were transformed into a corresponding fixed feature vector.

### 3.5.1 Keyboard Feature Vector

[7] uses a unique technique for transforming free-text data into fixed feature vectors. Their process relies on two critical aspects (1) the adjacency between two characters typed (2) the side of the keyboard where the two characters are typed. By combining these two components, a set of feature vectors that capture the underlying pattern of the sequence of keystrokes typed is produced. The first component, the adjacency between two characters, can be better visualized in the following figure.

The first letter typed in the figure is "G", and depending on the next character typed the adjacency can be different. For example, if "F" is pressed after, then there would be a first adjacency as the two characters are right next to each other. Moreover, if "I" was pressed instead, a third adjacency would exist, and so on for other characters. The second component is more straightforward, where each pair of

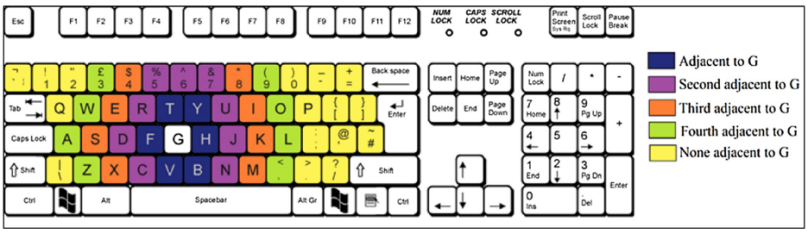


**Fig. 1.** Key-pair classification.

Figure 3: Keyboard Adjacency Map

characters typed can fall into one of the following categories:

- Left Left (LL) - Both keys pressed are on the left side of the keyboard
- Right Right (RR) - Both keys pressed are on the right side of the keyboard
- Both (B) - The two keys pressed are on different sides

Going back to the previous examples with the letter "F" and "I", if "F" is pressed after "G" then the resulting keyboard tag would be (LR), whereas "I" would produce a (B) tag. Their process produced 15 possible categories a key-pair can be mapped into, each of these categories containing five time-based features: Up-Down, Down-Up, Up-Up, Hold1, Hold2. Altogether, these 15 categories with five values in each category, are concatenated to produce a fixed-feature vector of length 75.

### 3.5.2 Mapping DKI

This paper follows a similar approach for mapping the DKI to fixed-feature vectors. Even though these mappings can be directly performed on the free-text data directly as it is processed, DKI were used as the input. We first captured all the unique pairs in the digraphs (non-zero values), saving each of the timestamps corresponding to a key-pair in a dictionary. An adjacency keyboard based on a MacBook was then constructed and traversed with a breadth-first search (BFS) to compute the level of adjacency for each pair of characters in the dictionary computed previously. However, unlike [7], there are only 4 adjacency levels used as to reduce the amount of empty (zero) features, these are 1,2,3, and None. Additionally, we modified the five time

based features captured to:

- Up-Down
- Down-Up
- Up-Up
- Down-Down
- Hold

Nonetheless, we use the same keyboard side tags, LL, RR, or B. From our process, a key-pair mapping can fall into one of 12 categories each containing five times based features, altogether creating a fixed feature vector of length 60 once all the categories have been concatenated. The table below lists all the possible categories a key-pair can be mapped into.

| Keyboard Side Category | Feature Set |
|---|---|
| 1st Adjacent Left Side | AL-H AL-UU AL-DD AL-UD AL-DU |
| 1st Adjacent Right Side | AR-H AR-UU AR-DD AR-UD AR-DU |
| 1st Adjacent Both Side | AB-H AB-UU AB-DD AB-UD AB-DU |
| 2nd Adjacent Left Side | SL-H SL-UU SL-DD SL-UD SL-DU |
| 2nd Adjacent Right Side | SR-H SR-UU SR-DD SR-UD SR-DU |
| 2nd Adjacent Both Side | SB-H SB-UU SB-DD SB-UD SB-DU |
| 3rd Adjacent Left Side | TL-H TL-UU TL-DD TL-UD TL-DU |
| 3rd Adjacent Right Side | TR-H TR-UU TR-DD TR-UD TR-DU |
| 3rd Adjacent Both Side | TB-H TB-UU TB-DD TB-UD TB-DU |
| None Adjacent Left Side | NL-H NL-UU NL-DD NL-UD NL-DU |
| None Adjacent Right Side | NR-H NR-UU NR-DD NR-UD NR-DU |
| None Adjacent Both Side | NB-H NB-UU NB-DD NB-UD NB-DU |

Table 1: Table of the different timing features

# CHAPTER 4

## Architecture

In this chapter, the paper will go more in-depth on the various classical and deep learning models used for binary classification and the corresponding hyperparameters, as well as the hyperparameter tuning involved for binary classification and generating data.

### 4.1 Binary Classification

The buffalo dataset consists of 148 users, each containing a several thousand keystrokes from doing the same tasks as the other users. Even though multi-classification can be leveraged for authenticating users rather than a single binary classification model, the implementation would be impractical for a commercial enterprise. Multi classification would involve more input from many different users, and the increased computational cost and added noise from more data used would produce a model that takes significantly longer to train and is less robust. Overall, binary classification is more practical for this problem as the main focus of these models is to uncover hidden typing patterns for a given user and detect whether these patterns exist in a given attempt - a binary problem. The goal of these models are to each take in a portion of a user's positive data and select limited negative samples from other users, and identify whether a given sample belongs to the user or an intruder. Furthermore, as positive data imbalance is a common issue when working with keystroke authentication, utilizing a Stratified K-Fold rather than the common K-Fold Cross Validation because the dataset is partitioned based on the proportion of one class to another. For example, a dataset with .3% positive and .7% negative would have a training and testing dataset with these same ratios.

## 4.2 Traditional Models

As mentioned earlier, this paper uses a novel feature engineering technique with influence from [7] that transforms the DKI from [1] to create fixed feature vectors traditional algorithms can train on. The paper uses these traditional algorithms as a baseline for comparison with other studies that use classical methods for free-text data, and to highlight the differences in performance between classical and deep learning models.

### 4.2.1 Hyperparameter tuning

In order to create powerful and robust models, a reasonable amount of hyerparameter tuning is undergone to find the parameters that significantly improve the model. This paper uses random grid search with the features in the table below to find the optimal set of initial parameters for each model. Moreover, as each architecture varies in the additional hyperparameters provided in the architecture, each model will have an additional set of architecture related parameters that also undergo a grid search.

The underlying principles and methodology behind KNN is explored in Section 2.4.1. This paper uses 3 parameters to optimize which are: the neighbors, the power parameter p, and weights. By using different neighbors in KNN, the model can find an optimal set of neighbors that preserves the local structure of the data without overfitting, and generalize the data well without underfitting. Moreover, experimenting with the power and weights parameter helps the model find a distance metric and weight distribution between neighbors that improves itself. The figure below highlights the different ranges for the parameters tuned and the resulting values.

| Parameter | Search Space | Result |
|---|---|---|
| neighbors | [10,20,30] | 20 |
| p | [1,2,3] | 1 |
| weights | uniform, distance | distance |

Table 2: Best hyperparameters for K-Nearest Neighbors

### 4.2.2 SVM

The underlying principles and methodology behind SVM is explored in Section 2.4.1. This paper uses the following three parameters to optimize: the regularization parameter C, the Kernel used, and the Gamma used. By modifying the C we can adjust values to find a nice balance between low testing and low training error. A similar process occurs with the gamma parameter to find a decision boundary that does not overfit the data too much. Lastly, experimenting with different kernels allows us to better understand which kernel best handles the data. The figure below highlights the different ranges for the parameters tuned and the resulting values.

| Parameter | Search Space | Result |
|---|---|---|
| C | [.01, .1, 1, 10] | 1 |
| Kernel | [3, 5, 7] | 3 |
| Gamma | [.001, .01, .1] | .1 |

Table 3: Best hyperparameters for Support Vector Machines

### 4.2.3 Random Forest

The underlying principles and methodology behind Random Forest is explored in Section 2.4.1. This paper uses 3 parameters to optimize which are: the number of estimators, the max features, the max depth, and the bootstrap. By exploring the data with different numbers of estimators, the performance can improve, but at a computational cost. Furthermore, using different thresholds for the number of features and varying the depth of the trees allow us to capture some control over

the randomness of the forest generated, striking an ideal balance between a simple forest (potentially underfitting) and a complex forest (potentially overfitting). The figure below highlights the different ranges for the parameters tuned and the resulting values.

| Parameter | Search Space | Result |
|---|---|---|
| estimators | [10,50,100] | 100 |
| max features | auto, sqrt | sqrt |
| max depth | [5,15,25] | 25 |
| bootstrap | [True, False] | True |

Table 4: Best hyperparameters for Random Forest

### 4.2.4 Logistic Regression

The underlying principles and methodology behind logistic regression is explored in Section 2.4.1. This paper uses the following three parameters to optimize: the number of estimators, the learning rate, and the bootstrap variable. By exploring the data with different numbers of estimators, the performance can improve at a computational cost. Furthermore, we use various learning rate parameters to find a balance between the rate of convergence and model generalization. The figure below highlights the different ranges for the parameters tuned and the resulting values.

| Parameter | Search Space | Result |
|---|---|---|
| estimators | [10, 100, 150] | 100 |
| learning rate | [.001, .1, 1] | .1 |
| bootstrap | [True, False] | True |

Table 5: Best hyperparameters for Logistic Regression

## 4.3 Deep Learning Models

Since deep learning models are capable of achieving accurate results that are comparable to traditional models as seen by [1] [5], this paper uses the deep learning

models from the former as a baseline for highlighting the quality of the generated data produced implemented.

### 4.3.1 Hyperparameter tuning

As mentioned previously this paper will be leveraging the deep learning models that align with the structure presented in [1]. The following hyperparameters used be viewed in the table below.

| Parameter | Result |
|---|---|
| epochs | 200 |
| learning rate | .01 |
| optimizer | Adam |

Table 6: Initial hyperparameters for CNN

Since the GAN architectures implemented were computationally expensive to train, we provided a limited search space for the initial parameters as seen in the table below that would be used for all GAN models. Additionally, we explore hyperparamter tuning within the varying GAN architectures to find the most robust version.

| Parameter | Search Space | Result |
|---|---|---|
| epochs | [50, 100, 150] | 50 |

Table 7: Initial hyperparameters for GANs

### 4.3.2 CNN

The underlying principles and methodology behind a CNN is explored in Section 2.4.1. This paper utilizes the (5x42x42) DKIs of each user as input for the CNN. The CNN itself consist of two convolutional layers, max-pooling layers, fully-connected layers, and a dropout layer, each working to help the model achieve a meaningful representation of the keystroke data without overfitting. Ultimately, producing a

probability (via sigmoid) of the likeliness a given sample is authentic or not. The entire architecture can be viewed in the image below.

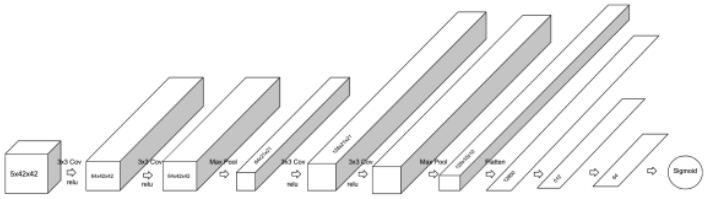


Figure 4: CNN Architecture

### 4.3.3 GAN

Varying GAN architectures such as WGAN, CGAN, and DCGAN are leveraged in this paper to demonstrate the efficacy of these deep learning models for producing quality positive data. Ultimately, we incorporate different architectures to identify how the baseline models handle generated data, and how the generated data from each GAN compares to one another. Generally speaking, each GAN utilizes the same underlying structure which consists of a generator and discriminator. These two networks are depicted in the figure below where the generator network is producing images that are supposed to be similar to the discriminator, and the discriminator is learning to distinguish the real and fake images. This results in an endless back and forth until the fake data is indistinguishable from the authentic data in this case. Since GAN architectures usually work with input of (64x64) and (128x128), we decided to pad the DKIs of (42x42) to produce a new set of DKIs that are (64x64) and can be used as input for GAN training. The next chapter will touch on how the generated data of 64x64 is tested with the baseline CNNs. This section will go in-depth about the various hyperparameters that were tuned and could be tuned in future works.

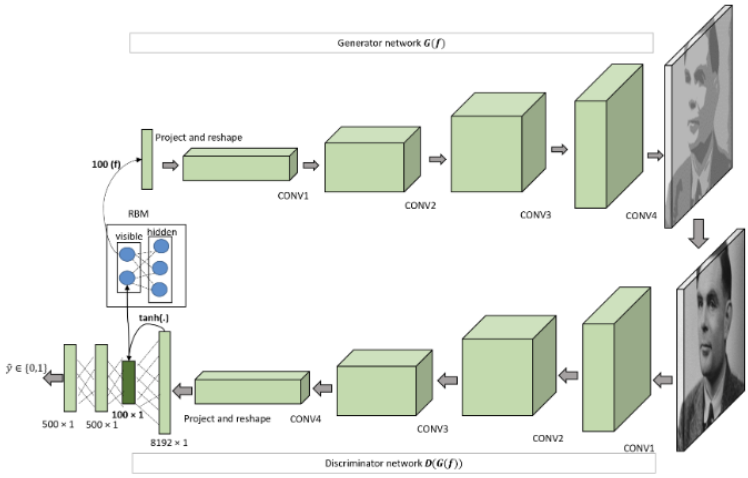


Figure 5: GAN Architecture

### 4.3.4 DCGAN

The underlying principles and methodology behind a DCGAN is explored in Section 2.4.1. The hyperparameters batch size, learning rate, were experimented with as other parameters required modifying the internal structure of the model. Furthermore, the batch size and learning rate are tuned in order to balance the trade-off between faster convergence and more accurate gradient updates during training, which can result in more stable and better quality synthetic data generated by the GAN model. The following GAN architecture hyperparameters available can be seen in the table below.

| Parameter | Search Space | Result |
|---|---|---|
| learning rate | [2e-6, 2e-5, 2e-4] | 2e-6 |
| batch size | [16, 32, 64] | 64 |

Table 8: Best hyperparameters for DCGAN

### 4.3.5 WGAN

The underlying principles and methodology behind a WGAN is explored in Section 2.4.1. For the WGAN hyperparameters, this paper experimented with the learning

rate, critical iterations, and weight clipping. The learning rate is straightforward to determine by finding a value that leads to optimal results without a significantly slow convergence,similarly with weight clipping and critic iterations. Overall, a reasonable amount of critic iterations, a small learning rate, and a tight weight clipping can lead to a stable training process that converges and produces quality data.

| Parameter | Search Space | Result |
|---|---|---|
| learning rate | [9e-5, 9e-4, 9e-3] | 9e-5 |
| critic iterations | [2, 5, 15] | 5 |
| weight clipping | [.01] | .01 |

Table 9: Best hyperparameters for WGAN

### 4.3.6 CGAN

The underlying principles and methodology behind a CGAN is explored in Section 2.4.1. Similar to the WGAN, and DCGAN, this paper will experiment with the learning rate, critic iterations, and batch size. The optimal values for the CGAN can be found in the table below.

| Parameter | Search Space | Result |
|---|---|---|
| critic iterations | [2, 5, 15] | 5 |
| learning rate | [1e-4, 1e-5, 1e-6] | 1e-5 |

Table 10: Best hyperparameters for CGAN

Moreover, it is worth noting that the CGAN architecture for this problem is unique in comparison to the other GANs. Unlike the other architectures, the CGAN also accepts class labels that are combined with the noise vector which helps generate data belonging to a certain class. The figure below illustrates the critical difference between the generator and discriminator with the original GAN architecture.

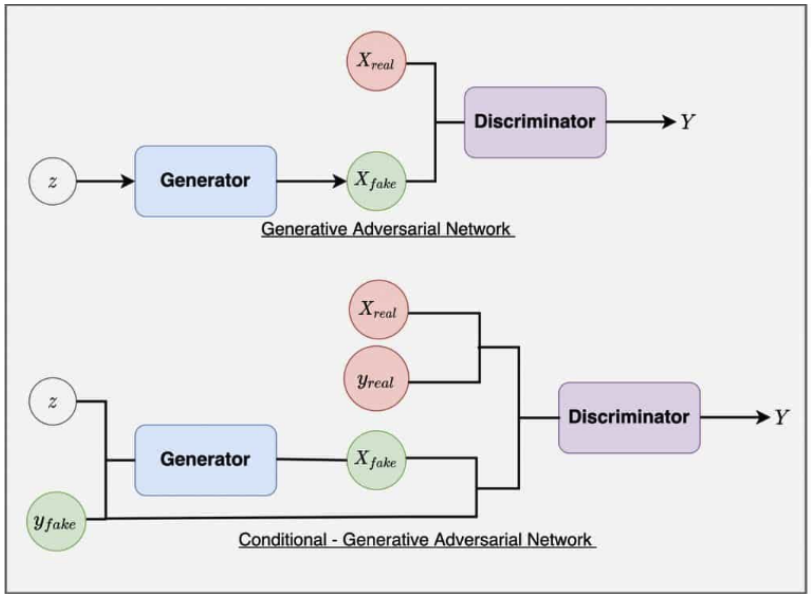


Figure 6: CGAN Comparison with Original GAN

# CHAPTER 5

## Experiments and Results

In this section this paper will retouch on the goals and intended objectives expected while conducting different experiments. Moreover, this chapter will be divided into two main sections (1) that evaluates the performance of traditional models on free-text data that has has undergone feature engineering (2) that explores performance of GANs for generating quality data, and the performance of CNNs trained with and without GAN data.

### 5.1 Experiment Strategy

As mentioned in the prior chapter these experiments will be conducted with free-text data that has undergone separate feature engineering techniques for deep learning and traditional models. This data is first used with traditional models to identify if any performance metrics are comparable to prior studies that also use traditional methods, and as another baseline for the deep learning models that will be implemented next. Secondly, the data is then fed to train CNN models that follow the same structure as mentioned in section and achieve a very low EER. Lastly, the original data the CNN has been trained is fed into the GANs which generates data that removes the padding and compares it with the CNNs. The overall expectations are that the traditional methods will be able to identify but will not outperform deep learning methods, and that GANs will be able to produce quality data that the CNNs can train on enhance their performance.

### 5.2 Experiment Metrics

With any machine learning model, in order to gain insight about the performance of the model and how it handles the data, evaluation metrics must be incorporated to provide different perspectives while assisting in identifying the strengths and weaknesses. This paper utilizes three common evaluation metrics and a biometric

evaluation metric. The three common metrics consist of accuracy, precision, and recall. Accuracy refers to the percentage of correctly identified predictions. However, this metric can be misleading, especially when there is data imbalance (as seen in our data), since the accuracy can be skewed to be high but impractical in a commercial setting. As a result, precision and recall are leveraged to gain more perspective about the model where accuracy falls short. Precision gives us the accuracy of the positive predictions while recall measures the completeness of the positive predictions. The three metrics previously discussed rely on a set of variables for constructing the formula: FN, FP, TP, TN. These variables are quite straightforward True Positive (TP) instances correctly identified, True Negative (TN) instances correctly identified, False Positive (FP) instances incorrectly identified, and False Negative (FN) instances incorrectly identified. Altogether, the variables previously mentioned that form the formulas for accuracy, precision, and recall can be seen in the figure below.

| | |
|---|---|
| Accuracy | $\textbf{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$ |
| Precision | $\textbf{Precision} = \frac{tp}{tp + fp}$ |
| Recall | $\textbf{Recall} = \frac{tp}{tp + fn}$ |

Figure 7: Traditional Evaluation Metrics

Moreover, the other biometric used is Equal-Error-Rate (EER) which refers to the point on the False Acceptance Rate (FAR) and False Rejection Rate (FRR) curves where the error rate is equal (i.e FAR == FRR). The figure below depicts the an example of where the EER might be for a model that has similar FAR and FRR curves. For this example if the EER rate is 5%, then this implies that adjusting the

threshold at which points are identified could result in a FRR and FAR of 5%. As a result, 95% of intruders and authentic users are identified correctly.

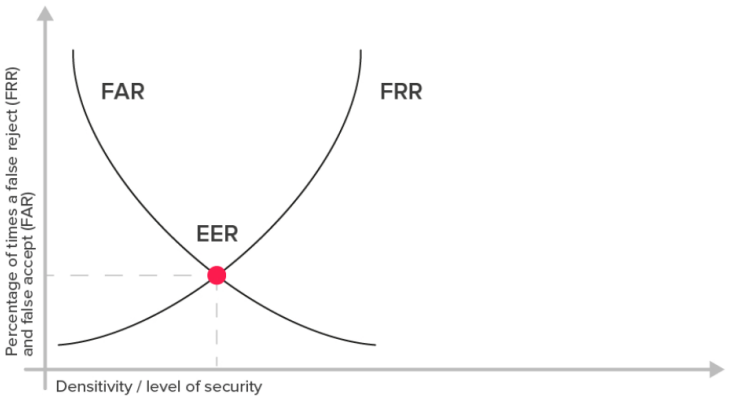


Figure 8: Biometric Evaluation Metric

## 5.3 Traditional Machine Learning Methods

This section will discuss the traditional machine learning methods leveraged for dynamic keystroke authentication and the accompanying results.

### 5.3.1 Results for free-text

As mentioned previously, the traditional methods were trained and tested on input data which consists of DKIs mapped into fixed feature vectors of length 60. Moreover, while the general accuracy of the models are relatively well as seen in the figures below, using other metrics such as the precision and recall uncover a different story about the performance of the models. When evaluating the models through precision and recall, the results clearly show that models are capable of identifying the intruder (high precision) but are unable to identify the correct user (low recall). For models evaluated with data consisting of keystroke sequences of 50 keystrokes, traditional models such as SVMs and KNN produce a great accuracy and a low FAR but a high EER, whereas logistic regression with boosting and Random Forest provide

an even greater accuracy, a low FAR, and a relatively low EER. Overall, random forest outperforms everyone with an accuracy of 95.74% and a EER 13.57%. These results indicate that with more fine-tuning to reduce the FNR and EER, a commercially feasible product capable of generalizing a user's typing patterns to a high degree of accuracy is possible.

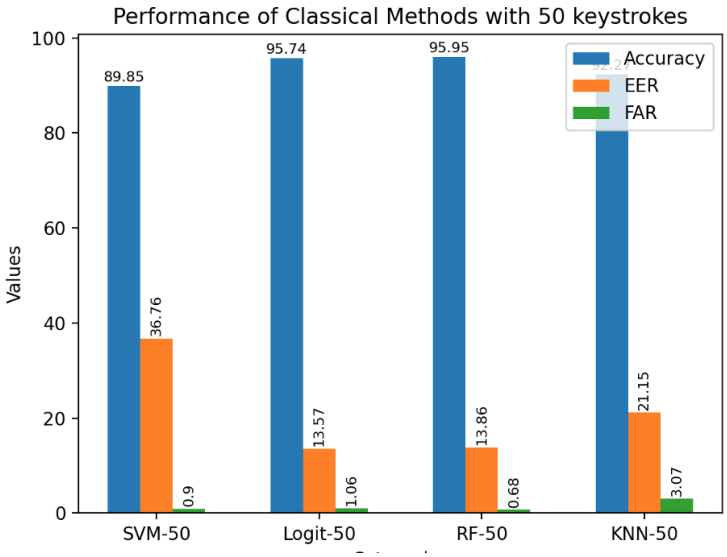


Figure 9: Models Performance - Classical Methods - 50

Moreover, the performance of the models seen in figure 4 contrasted with those in figures 5 and 6 illustrate how adding more data before producing the fixed-feature vectors for training can be an obstacle. As seen in the results from figures 5 and 6, as fixed-feature vectors are based on longer keystroke sequences like 75 and 100, the inclusion of more data leads to a poorer performance in a model's ability to distinguish an authentic user from an intruder. In comparison to figure 4, the more data is included the higher the EER and FAR increase, eventually leading to a KNN with a 4% FAR and an ERR of 68%. The results suggest that the addition of more keystrokes introduces more variation and complexity into the data, resulting in more complex data that the models cannot construct an accurate generalization of.

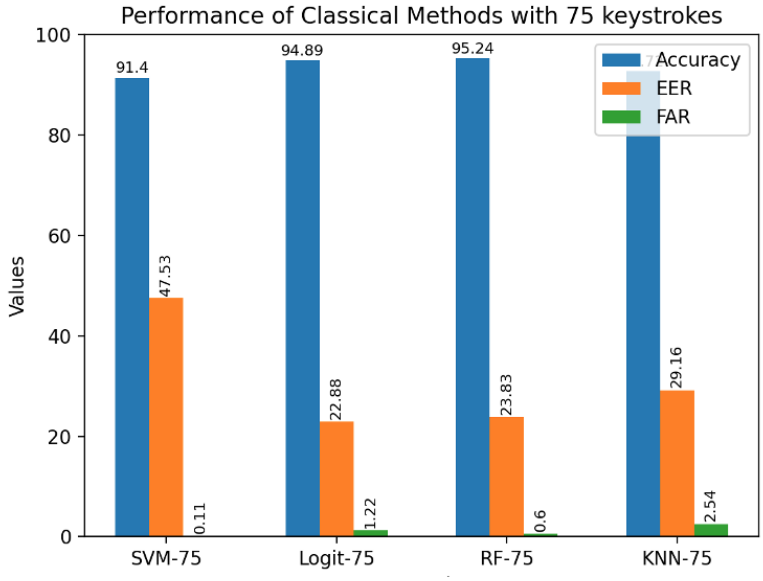


Figure 10: Models Performance - Classical Methods - 75

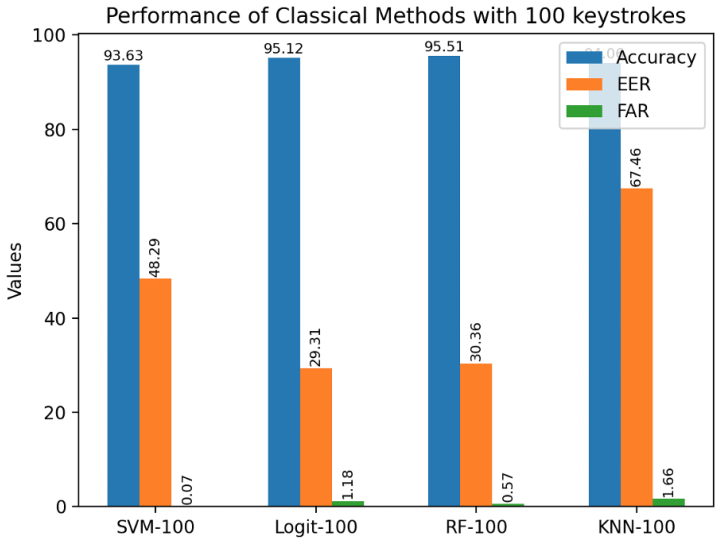


Figure 11: Models Performance - Classical Methods - 100

While these model do under perform when contrasted to the performance of the deep learning methods implemented, as the paper will discuss in the next section; the EER results of the best model are comparable to other studies that also measured the EER of their models on free-text data.

## 5.4 Deep Learning Methods

This section will cover more depth about the performance of the varying CNN architectures on DKIs derived from the free-text dataset and from the GANs. Additionally, the quality of the generated data and its impact on CNN architectures will be explored.

### 5.4.1 Results of CNNs of dataset

As mentioned previously, the CNNs constructed were based off the architectures in [1] and trained / tested using the same methodology. Moreover, results of these classifiers align with those seen in the prior research. Highlighted in the figure below, the CNNs trained on different keystroke sequence lengths achieve a high accuracy and considerably low EER when contrasted to the traditional models. Even though the results of all the CNNs are relatively similar, the CNN based on keystroke sequences of length 75 achieve an average accuracy of 88% and an EER of .09%.

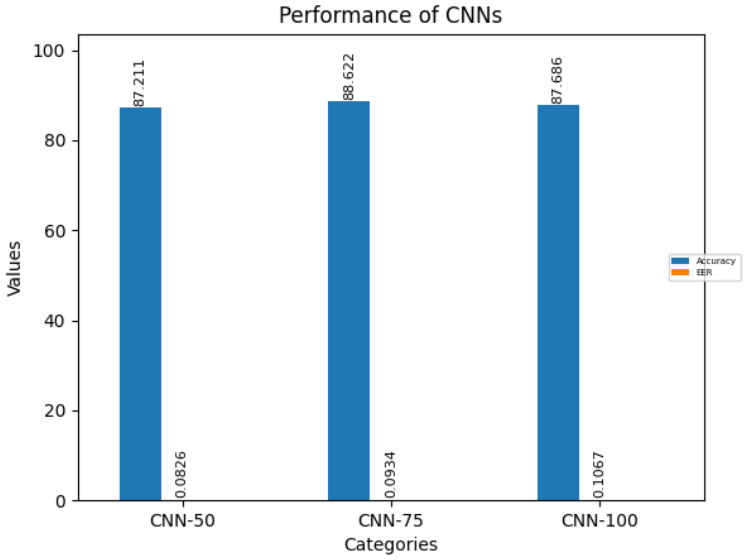


Figure 12: Classifier Performance - CNN

### 5.4.2 Results for GAN data and CNN

Several data generation experiments were conducted on different GAN architectures to identify any key advantages / disadvantages for data generation and for data quality. Moreover, the experiments produced a set of generated data (DKIs) that were then unpadded and evaluated with positive data on the trained CNNs in the prior section. The results of evaluating the CNNs with positive data, GAN data, and both (augmented data) on a keystroke sequence of 50, are depicted figures below. The GAN data is labeled as 1 for these experiments.

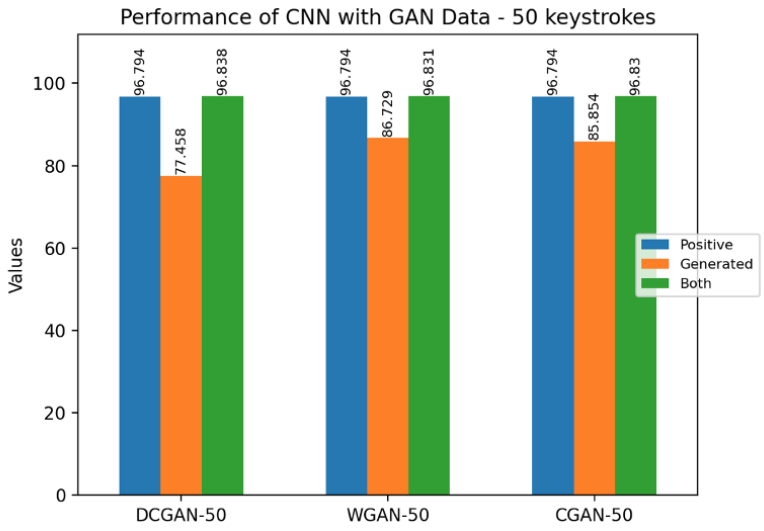


Figure 13: GAN Performance - 50

The figures demonstrate the impact varying GAN images produce on the classifier. When it comes to keystroke sequences of 50 and 100, the results are relatively similar with WGAN and CGAN producing images based on each individual user. On average the CNNs identified the generated images with 86% accuracy. As intended each classifier that is fed its positive data with negative data produces high accuracy since the positive data is the data it was trained with. Interestingly, when the CNNs are fed positive data that consists of the real and GAN data, the results do show that

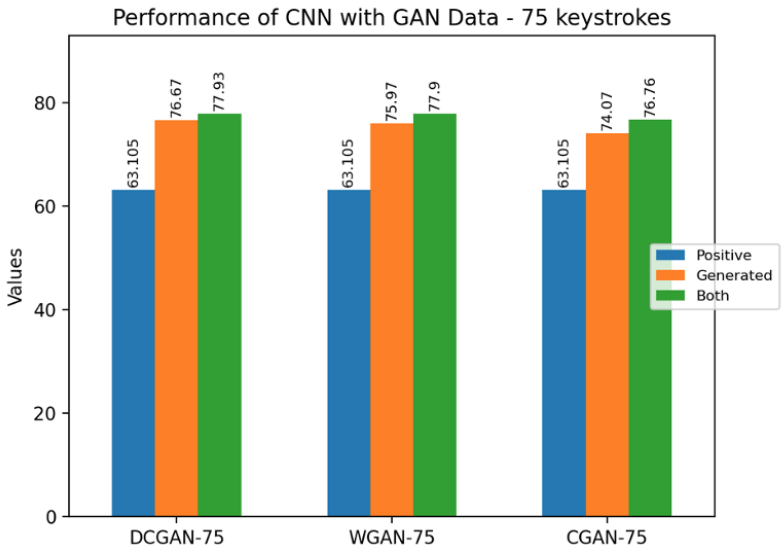


Figure 14: GAN Performance - 75

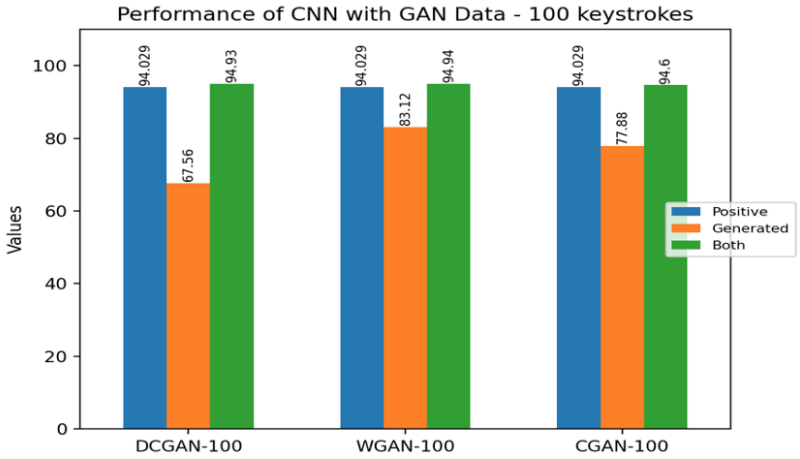


Figure 15: GAN Performance - 100

there is a slight increase in the accuracy as seen by the WGAN which improves the accuracy from 94% to 94.6%. Even though the increase is within a 1%, the results suggests classifier enhancement is possible. However, more experiments must be taken to ensure the quality of the data generated as will be seen in the following section.

### 5.4.3 Results for GAN data vs CNN

To provide more perspective about the quality of the data being generated, more experiments were conducted which essentially invert the ones presented in the prior section. The GAN data will be labeled 0, and the performance of CNN are recorded. The results of the CNN against actual negative data, and GAN data are illustrated in the figure below.

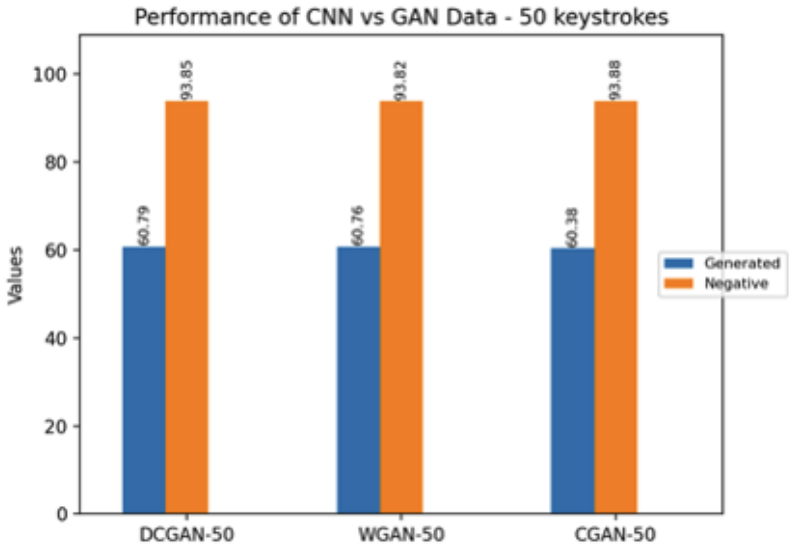


Figure 16: GAN Performance - 50

The results in the figure highlight that the CNNs perform poorly when treating the GAN data as separate, as seen in figure above with accuracy of 60.79%. Accuracy this low suggests that the data generated bears resemblance to the input data for a given user. As a result, these classifiers struggle to distinguish between the real data and generated data when treated separated. Moreover, the CNN with actual positive data against negative data performs with similar accuracy to those presented in [1].

### 5.4.4 Results for GAN + Real data vs CNN

The last experiments tested treat the GAN data at valid and compare it with the negative data. Moreover, the first test uses only GAN data and negative samples from

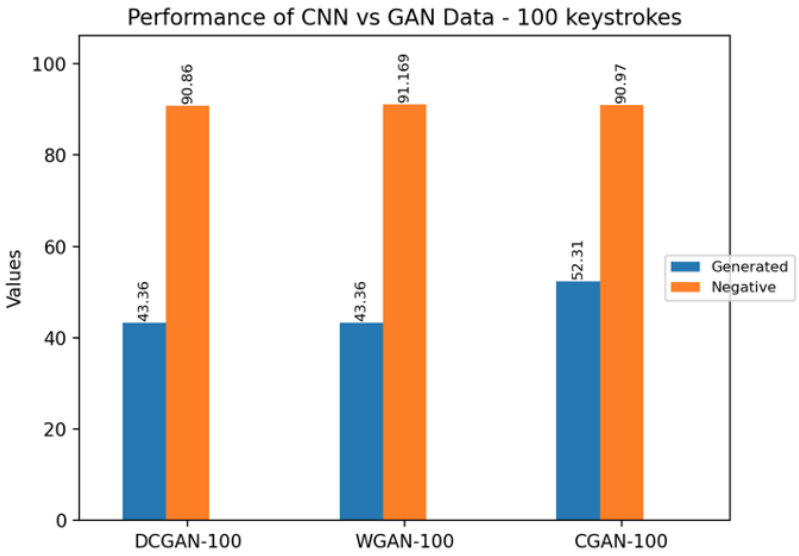


Figure 17: GAN Performance - 75

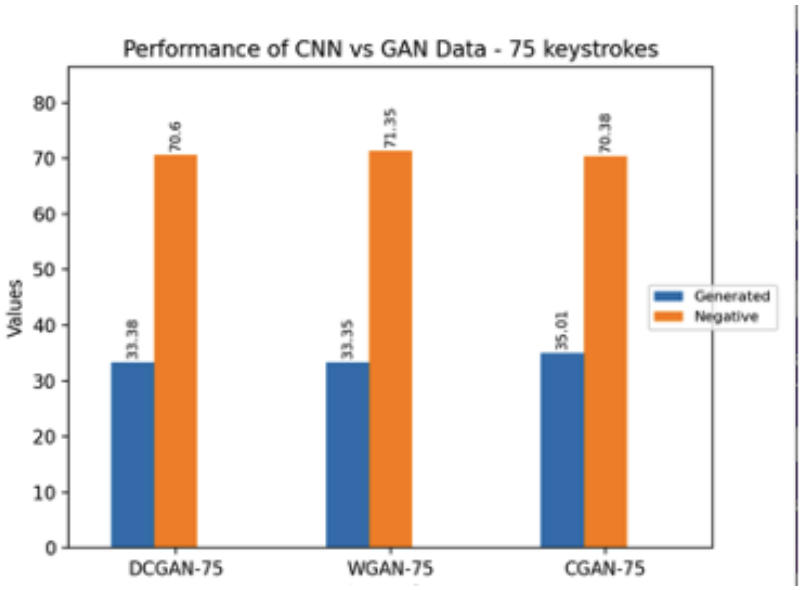


Figure 18: GAN Performance - 100

other users, while the second test combines both real and generated data (augmented data) and also compares it against the negative samples. The accuracy of each test is highlighted in the figure below.

As anticipated when the CNN uses GAN data as valid vs. negative samples,

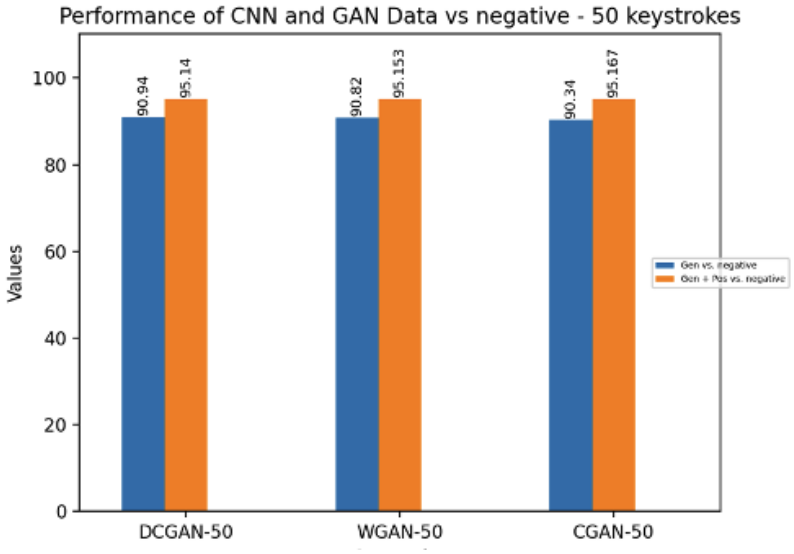


Figure 19: GAN Performance - 50

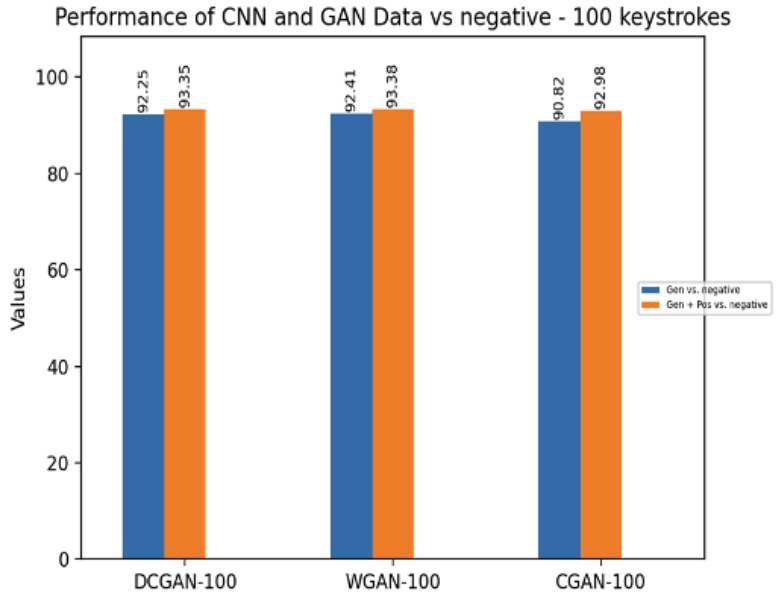


Figure 20: GAN Performance - 100

the performance suffers by a negligible amount. Figure - illustrates this point as the DCGAN data achieves an accuracy of 90.827% when evaluated with the CNN in contract to ann accuracy of 93.827% for positive vs.negative data in figure -. In other cases the accuracy actually increases slightly as seen WGAN in figure - with

92.411% vs 91.169%. Moreover, the second experiments produces an accuracy similar to those when the classifiers are tested against only positive data, and positive data vs. negative data. These results are depicted in the figures above where the accuracy differences are very minimal, emphasizing how well the data generated fits into a classifier trained on the data it was generated on. Since, the generated DKIs were able to perform well when evaluated by the CNNs, the next step was to retrain the CNNs with the GAN generated user data labeled as positive, following the same process as those in the first section.

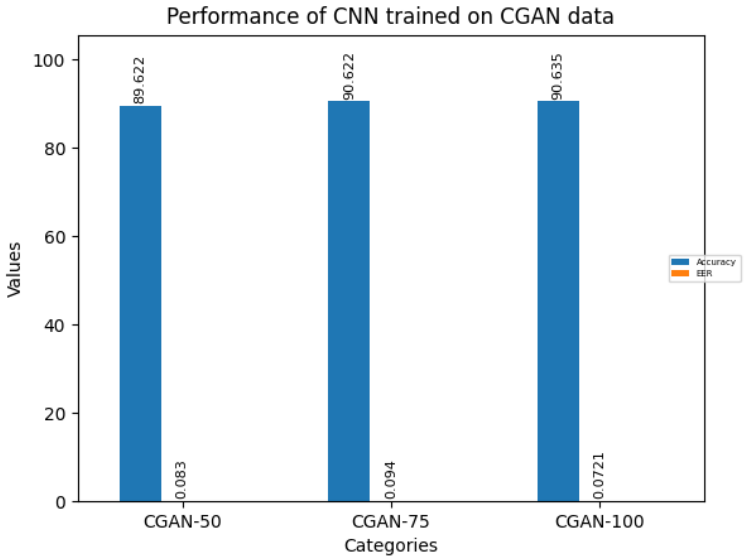


Figure 21: Models Performance - Classical Methods - 100

### 5.4.5 Results for CNN trained with GAN Data

The last step to verify that effectiveness and quality of the DKIs generated by the GANs was to retrain the CNNs. The classifiers at this step, in contrast with those in the first section, leverage the GAN samples by also including them as positive data. For the sake of ease, the CNNs for keystroke sequences 50, 75, and 100, were tested with only one architectures generated data for different sequences. Since CGAN

performed the best overall, the results presented in the figure below were based only on the CGAN data.

### 5.5 Discussion

The results retrieved from all the experiments conducted produces a variety perspective regarding the performance of classical vs. deep learning methods in user authentication, and the utility of GANs to generate quality data while enhancing the deep learning methods. When contrasting classical and deep learning methods, the results clearly show that traditional methods can achieve a relatively high accuracy with a reasonable FAR and EER as seen by the Boosted Logistic Regression model on 50 keystrokes. However, these results obtained pale in comparison to the CNNs that can achieve a greater accuracy with a lower EER, but these deep learning models do still have an issue with the FNR. Moreover, when introducing GANs to resolve the positive data bottleneck by creating synthetics data resembling user data, the results illustrate the effectiveness. The GAN data by itself when evaluated produces a relatively reasonable accuracy around 85%. Furthermore, figures 16 - 18 demonstrate that treating the generated data as negative throws the classifier off producing low accuracy, indicating that the generated data has resemblance to the user data it was trained on. Finally, using the augmented data consisting of positive and generated data, and retraining the CNNs results in slightly higher accuracy with an EER with a negligible move in either direction. As mentioned before, this paper builds upon the deep learning classifiers presented in [1], we demonstrate that the CNNs and data used in the study can be further extended to generate more positive data and achieve a greater accuracy and lower EER.

# CHAPTER 6

## Conclusion

As technology continues to evolve and become more integrated with day-to-day life, the concern for security breaches (hacks) across multiple mediums increases. As a result, methods for incorporating and enhancing keystroke authentication must be implemented to reduce the likelihood of hackers gaining unauthorized access. This paper presents a comprehensive study utilizing traditional machine learning and deep learning methods in dynamic keystroke authentication, and demonstrates the potential of GAN techniques to create synthetic data to enhance the performance of CNN classifiers. Moreover, as byproduct of successfully generating quality data, the positive-data bottleneck presented in a data-driven commercial environment is further mitigated..

The paper trains classical machine learning models using DKIs based of varying sequence lengths that are transformed into fixed-feature vectors through feature engineering influenced by [7]. While the traditional models do produce high accuracy and reasonable EER in certain instances, these results still present an issue for the authentic user and do not exceed the performance of the baseline CNNs. Moreover, the CNNs in this paper follow the same architecture and feature engineering process as those presented in [1], receive similar results from that study. Emphasizing that deep learning models carry a higher capability for recognizing patterns from time-based features and can provide a better generalization. The DKIs generated from different GANs, are examined through different tests with the baseline CNN and are finally used to retrain the CNNs. Overall, the synthetic user data generated resembles the structure of those presented in the real dataset as is highlighted by the CNNs performance enhacned after retraining and the CNN struggling to identify the GAN data treated as negative before retraining.

Eventhough this paper focuses on generating data based on time-based features from dynamic keystroke dataset, the input data does not have to be limited to time features. Touch-based features are another form of data that captures unique, mostly static, individual characteristics like a user's touch area and pressure when typing. Furthermore, as CGANs requires additional conditional input to help construct more precise images, dynamic keystroke authentication can be performed on a dataset with touch-based features utilizing the touch-based metric as conditional input to generate data more precise with the user's time and touch patterns. Overall, this study contributes to the growing body of literature on the use of GAN techniques for data generation and highlights their potential to revolutionize the field of biometric authentication and other data-driven applications.

## LIST OF REFERENCES


[1] J. Li, H.-C. Chang, and M. Stamp, ‘‘Free-text keystroke dynamics for user authentication.’’ [Online]. Available: https://arxiv.org/abs/2107.07009

[2] G. E. N. Forsen, ‘‘Personal attributes authentication techniques.’’ [Online]. Available: https://ntrl.ntis.gov/NTRL/dashboard/searchResults/titleDetail/ADA047645.xhtml

[3] Killourhy, K. S., and R. A. Maxion, ‘‘Comparing anomaly-detection algorithms for keystroke dynamics,’’ pp. 125--134, 2009.

[4] L. de Marcos, J.-J. Martínez-Herráiz, J. Junquera-Sánchez, C. Cilleruelo, and C. Pages-Arévalo, ‘‘Comparing machine learning classifiers for continuous authentication on mobile devices by keystroke dynamics,’’ *Electronics*, vol. 10, no. 14, p. 1622, 2021.

[5] Y. Muliono, H. Ham, and D. Darmawa, ‘‘Keystroke dynamic classification using machine learning for password authorization,’’ *Procedia Computer Science*, vol. 135, pp. 564--569, 2018.

[6] J. Li, H. Chang, C. Wu, and M. Stamp, ‘‘Machine learning and deep learning for fixed-text keystroke dynamics,’’ *Advances in Information Security*, vol. 54, no. 2, pp. 309--329, 2021.

[7] A. Alsultan, K. Warwick, and H. Wei, ‘‘Improving the performance of free-text keystroke dynamics authentication by fusion,’’ *Applied Soft Computing*, vol. 70, pp. 1024--1033, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494617306816

[8] L. Xiaofeng, Z. Shengfei, and Y. Shengwei, ‘‘Continuous authentication by free-text keystroke based on cnn plus rnn,’’ *Procedia Computer Science*, vol. 147, pp. 314--318, 2019.

[9] A. Radford, L. Metz, and S. Chintala, ‘‘Unsupervised representation learning with deep convolutional generative adversarial networks,’’ 2016.

[10] M. Arjovsky, S. Chintala, and L. Bottou, ‘‘Wasserstein gan,’’ 2017.

[11] M. Mirza and S. Osindero, ‘‘Conditional generative adversarial nets,’’ *CoRR*, vol. abs/1411.1784, 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[12] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, ‘‘Generative adversarial networks,’’ 2014.

[13] L. Chu-Hsing, L. Jung-Chun, and L. Ken-Yun, ‘‘On neural networks for biometric authentication based on keystroke dynamics,’’ *Sensors and Materials*, vol. 30, no. 3, p. 385–396, 2018.

[14] J. Kim and P. Kang, ‘‘Freely typed keystroke dynamics-based user authentication for mobile devices based on heterogeneous features,’’ *Pattern Recognition*, vol. 108, p. 107556, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320320303599

[15] M. Antal, L. Z. Szabó, and I. László, ‘‘Keystroke dynamics on android platform,’’ *Procedia Technology*, vol. 19, pp. 820--826, 2015, 8th International Conference Interdisciplinarity in Engineering, INTER-ENG 2014, 9-10 October 2014, Tirgu Mures, Romania. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S221201731500119X

[16] H. Saevanee and P. Bhattarakosol, ‘‘Authenticating user using keystroke dynamics and finger pressure,’’ pp. 1--2, 2009.

# APPENDIX

## Zorak Likes Beans

### A.1 Oh Yes He Does

Appendices can have sections and subsections and so on.

### A.2 Really

Sections, subsections, or whatever should come in pairs.