BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

LAPTOP TOUCHPAD PALM DETECTION WITH AI/ML

DETEKCE DLANĚ NA LAPTOP TOUCHPADU S POUŽITÍM AI/ML

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MARK ALEXANDER MENŻYŃSKI**
AUTOR PRÁCE

SUPERVISOR **Prof. Ing., Dipl.-Ing. MARTIN DRAHANSKÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2020/2021

Bachelor's Thesis Specification

20593

Student:	**Menzyński Mark A.**
Programme:	Information Technology
Title:	**Laptop Touchpad Palm Detection with AI/ML**
Category:	Artificial Intelligence

Assignment:

1. Study how to collect the data for AI/ML training, what is the nature of the data and implement tools for recording those data.
2. Study the literature about Synaptics touchpad and AI/ML techniques. Study what AI/ML techniques could be applied.
3. Decide on what AI/ML models can be used to detect unwanted touches on the touchpad based on the data from the Synaptics device. Analyze how those models work based on prototypes and how they are effective for the problem. Propose a model that will work best.
4. Implement the proposed model chosen in point 3.
5. Test, summarize, and compare results with existing solutions.

Recommended literature:

- HUANG, Tai-sou; HSU, Chun-Ming. *Touchpad module*. U.S. Patent No 10,684,705, 2020.
- OU, YangKun. User preference and usability assessments of touchpad surface tactile in laptops. *Human Factors and Ergonomics in Manufacturing & Service Industries*, 2020.

Requirements for the first semester:

- Items 1 and 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Drahanský Martin, prof. Ing., Dipl.-Ing., Ph.D.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	November 1, 2020
Submission deadline:	July 30, 2021
Approval date:	November 11, 2020

Abstract

The situation about palm rejection for laptops is less than ideal. Most research focuses on touchscreens, and there is minimal research on touchpads. Some research is possibly done privately in laptop manufacturer companies, but the technology is lacking behind regardless. This thesis explores several shallow and deep machine learning models and finds their accuracy to be very much sufficient. In addition, a real-time proof of concept is implemented to demonstrate the model's capabilities.

Abstrakt

Situace ohledně detekci a odmítnutí dlaně na laptopech je méně než ideální. Většina výzkumů se zabývá odmítnutím dotyků na dotykových obrazovkách, a na laptopy probíhá téměř žádný. Patrně nějaký uzavřený výzkum probíhá uvnitř výrobců laptopů, ale i přes to je technologie pozadu. Tato práce prozkoumává několik metod plytkého a hlubokého strojového učení, a výsledná přesnost byla zjištěna jako více než dostačující. Také implementuje aplikaci v reálném čase na demonstraci modelu.

Keywords

touchpad, palm, touch, rejection, machine learning, neural network, deep learning, random forest, support vector, logistic regression

Klíčová slova

touchpad, dlaň, dotyk, odmítnutí, strojové učení, neuronové sítě, hluboké učení, náhodný les, podpůrné vektory, metoda podpůrných vektorů, logistická regrese

Reference

MENŻYŃSKI, Mark Alexander. *Laptop Touchpad Palm Detection with AI/ML*. Brno, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Prof. Ing., Dipl.-Ing. Martin Drahanský, Ph.D.

Rozšířený abstrakt

Tato bakalářská práce se zabývá zkoumáním modelů strojového učení u problému detekce dlaně u laptopů. Modely, které byly prozkoumány, jsou: umělá neuronová síť, konvoluční neuronová síť, rekurentní neuronová síť, logistická regrese, model podpůrných vektorů a náhodný les. Problém detekce dlaně se týká případů, kdy se uživatel dotýká touchpadu (dotykové plochy pod klávesnicí laptopů), bez záměru s touchpadem interagovat, což způsobuje nechtěný vstup na počítači, jako pohyb kurzoru, klikání kurzorem nebo některé ovládací gesta, jako například minimalizace všech oken na obrazovce a tyto případy následně detekovat za účelem zamezení těchto vstupů. Což bez diskuze dělá používání počítače se zapnutým touchpadem příjemnější. Tato práce rovněž prozkoumává, jak může touchpad ovlivňovat věci i z ergonomického pohledu.

Cílem této práce je poukázat na aktuální nevyspělost a zastaralost problému detekce dlaní na touchpadech. Současně v případě tabletů se provádí značné množství výzkumů a na laptopy se zapomíná. Při tom laptopy jsou početnějším zařízením, každý laptop touchpadem disponuje, a některé, ačkoliv malé množství, dokonce neumožňují jeho vypnutí. Dále je cílem předvést modely, které by pro tento problém byly vhodné. Což, doufejme, bude motivovat společnosti, které laptopy vyrábí, aby k tomuto problému detekce dlaně přistupovala s větší iniciativou.

Zaměřil jsem se dosáhnutí přesnosti modelů vyšší než jaká je konkurence u dotykových obrazovek, protože ohledně touchpadů nejsou k dispozici výzkumy. Dalším původním zaměřením bylo dosáhnout nižší časové a paměťové náročnosti, ale nepodařilo se mi najít výzkum, který by toto zkoumal u konkurenčních řešení dotykových obrazovek, a proto tyto výsledky nebyly porovnány. Během práce na této práci se také objevil problém, který v podstatě zamezuje implementování reálného odmítnutí dotyků. Problém je způsoben tím, jak jsou laptopy navrženy. Touchpad je připojen velmi pomalou sběrnicí a sám o sobě funguje jen jako oddělená periferie, než jako senzor laptopu. Z tohoto důvodu touchpad nikdy nebyl připraven na množství dat, které sběrnicí prochází při kolekci nezpracovaných dat. To způsobuje, že při sběru nezpracovaných dat dochází k nereakčnosti touchpadu. To znamená, že pohyby po touchpadu často nejsou ve stoprocentním množství interpretovány počítačovým systémem pro pohyb kurzorem, což omezuje spoustu výzkumů. Tato práce se snaží tento problém obejít.

Nejvyšší přesnost detekce dlaně, která byla dosažena, byla metodou náhodných lesů, kde při předem naměřené sbírce dat, odlišné od trénovací sbírky dat, byla naměřena přesnost přes 99.5 %. Nutno ale uvést, že jak trénovací, tak testovací data nejsou dobrou reprezentací tohoto problému, protože byly nasbírány jen mnou, bez výzkumu kvality dat. Byla také vytvořena aplikace pro demonstraci tohoto modelu, která na obrazovce zobrazuje aktuální data z touchpadu a uvádí, jestli by se měla tato data zamítnout nebo ne (jestli se jedná o dotyk prstem a nebo dlaní). Ukázka tohoto programu ve video podobě je součástí odevzdané práce.

Laptop Touchpad Palm Detection with AI/ML

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of prof. Martin Drahanský, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Mark Alexander Menżyński
July 30, 2021

Acknowledgements

Foremost, I would like to express my thanks to my supervisor Prof. Ing., Dipl.-Ing. Martin Drahanský Ph.D. for supervising this thesis.

I would also like to thank my external advisor Kevin Martin, and consultant Karol Herbst for the professional and nonprofessional help. Also, thanks to Benjamin Tissoires, for expertise with the touchpad device and its driver.

Thanks also to my family, colleague Carlos Soriano Sanchez, university colleagues Betka Štajerová and Ondřej Kinšt for their moral support.

Lastly, I want to express my gratification towards Red Hat, for adopting this thesis as part of my internship.

Contents

1 Introduction **3**

2 Literature review **4**

2.1 Training data . . . 4

2.2 Existing solutions . . . 4

2.2.1 Hardware solutions . . . 4

2.2.2 Software solutions . . . 6

2.2.3 Software machine learning solutions . . . 7

2.3 Machine learning models . . . 8

2.3.1 Learning approaches . . . 9

2.3.2 Gaussian Mixture Model . . . 10

2.3.3 Linear regression . . . 12

2.3.4 Logistic regression . . . 12

2.3.5 Support Vector Machine . . . 12

2.3.6 Random Forest . . . 14

2.4 Artificial neural networks . . . 15

2.4.1 Perceptron . . . 16

2.4.2 Cost Function . . . 17

2.4.3 Gradient Descent . . . 17

2.4.4 Back-propagation . . . 17

3 Implementation **19**

3.1 Data collection . . . 19

3.1.1 Raw data capture . . . 19

3.1.2 Data elimination . . . 20

3.1.3 Data augmentation . . . 23

3.2 AI/ML prototyping . . . 27

3.2.1 Deep learning models . . . 28

3.2.2 Shallow learning models . . . 29

3.3 Proof of concept . . . 32

4 Testing and summarization **34**

4.1 Dataset . . . 34

4.2 Neural Networks parameters . . . 35

4.2.1 Artificial neuron networks . . . 35

4.2.2 Convolutional neuron networks . . . 36

4.2.3 Recurrent neural networks . . . 36

4.3 Prototypes analysis . . . 37

4.3.1 Accuracy 37
4.3.2 Time complexity 38
4.3.3 Memory complexity 38
4.4 Summarization 38

5 Conclusion **40**

Bibliography **41**

Chapter 1

Introduction

Palm rejection is a technology broadly researched for tablets, and other touch screen devices, especially those using styluses. But touchpads on laptops tend to lag behind. There is very little available research about laptop touchpads, especially for touch rejection, and access to the touchpad itself as a device is usually minimal. This thesis aims to raise consciousness about current issues with touch rejection on laptops, motivate companies to focus more on touch rejection, and inspire more research toward touch rejection on laptops.

A touchpad, also called „trackpad,“ is an input module that allows the user's finger on a smooth panel to control a device's cursor, as well as other tasks, using gestures. Almost every laptop features one, and they tend to be placed between the keyboard and the user, usually directly under the space bar. Using a touchpad can also free up workspace room otherwise used by a computer mouse, reducing the required workspace while still maintaining productivity. Time and muscle movement when switching between a keyboard and pointer device is also reduced [20]. However, people tend to prefer using a computer mouse. Furthermore, the wrist extension angle tends to be higher, as shown in study [19].

Palm rejection is a technology that ignores touches made by a palm, which allows the user to use the touchpad as a palm rest without activating any cursor movement. Research by Camilleri et al. [7] found an association between palm rejection and more significant wrist extension, lesser discomfort, and faster task speeds. On touch screens, palm rejections seem to help lower hand positions. Lower hand positions, according to research from Erdelyil et al. [10] decrease the shoulder movement and shoulder muscular effort, and discomfort.

The palm rejection process analyzes movements on a device's touch sensor, classifying the movement whether it is a palm and rejecting input assumed to be a palm. This process eliminates inputs that the user probably did not want to use to control the device. This thesis aims to use artificial intelligence machine learning techniques to implement palm rejection for a laptop's touch device. Those techniques have not been applied broadly for palm rejection yet by the public. Suppose any of the explored AI techniques appeared to be either more accurate, faster, or less resource-intensive. In that case, it could become an alternative to current solutions and could replace or extend current solutions of palm rejection on laptops. The thesis' goal is to find a solution that would be better in at least one of those three criteria mentioned and learn about machine learning algorithms and their behaviour, their uses in practice.

Chapter 2

Literature review

2.1 Training data

This thesis is created with data extracted from a Synaptics touchpad driver RMI (Synaptics Register Mapped Interface). And is developed on a Fedora system with kernel version `5.9.0-rc5`. Training data are collected by capturing the V4L2 stream of a touch device stream. V4L (Video4Linux) is an API and collection of drivers used for video stream in Linux systems.

The current way of capturing data has multiple limitations. The first limit is that resolution of captured data is low. Using a Synaptics touchpad in a Lenovo T580 laptop, debug data resolution is 20 × 13 „antennas.“ Such resolution is very low for many AI techniques, and overcoming this resolution is one of the most considerable obstacles of this work. Each antenna has a value describing the pressure applied; a higher-touch surface tends to have higher values. The second substantial limit is the bandwidth that a bus connecting the touchpad can put through.

The possibility that with current resolution will not be sufficient for training ML models, an approach to upscale the data and approximate them will likely be required to have satisfactory results. Most touches appear to be round; thus, the idea is to approximate data to circles. Different approaches, such as using AI techniques for increasing detail, should be explored too.

2.2 Existing solutions

For laptop touchpads, existing solutions are unfortunately proprietary and not documented. Drivers get already processed data, so it is challenging to explore open-source drivers. Nevertheless, touchscreens are broadly explored and documented. Thus, this chapter will mainly aim at current solutions for touch screens.

2.2.1 Hardware solutions

Hardware solutions require additional equipment to function correctly. However, when performed right, they tend to be reliable. This subsection will cover few hardware solutions that could be used.

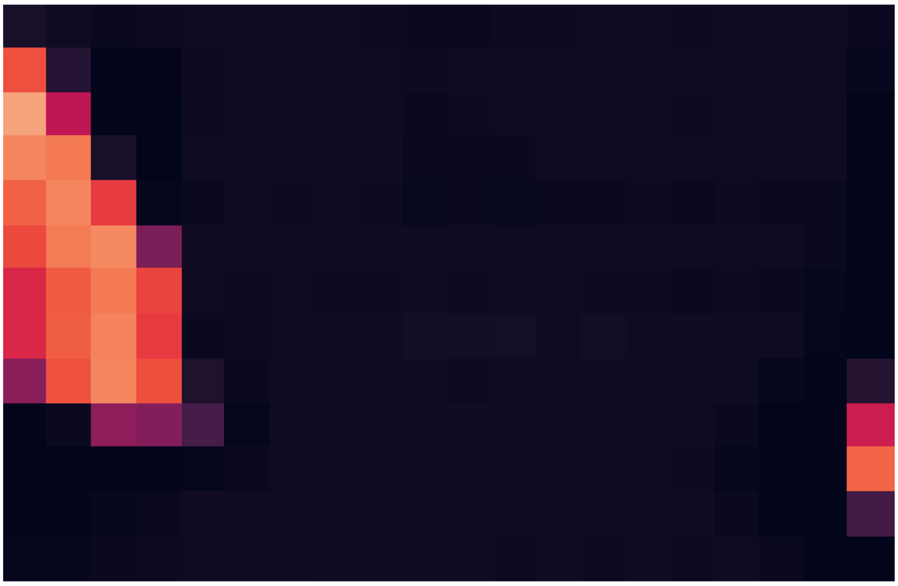


Figure 2.1: Example of raw data interpreted as a heatmap.

Active stylus

Devices with touch screens that aim to use an active pen tend to have special hardware detecting the stylus, such as ultrasonic sensors sensing ultrasonic pulses from a pen. Other approaches are using infrared light or pen sending pressure data over Bluetooth. Another approach, which can be seen, for example, in Microsoft Surface, or Samsung Galaxy Note, uses resonance inductive coupling or sensors for sensing the magnetic tip of the pen. It is also easier to differentiate the pen from the palm because it has a very uniform shape and capacitive properties. However, laptop touchpads do not use pens. Those methods are mostly not used; fingertips are used instead, which are not so uniform and do not always have the same capacitive properties as a dedicated pen.

Spacer dots

There is also a palm rejection approach that does not use any additional modules, but the approach is still a hardware solution. The approach uses the setting distance of spacer dots. Spacer dots create spacing between two conductive layers of the touchpad [23]. The distance of those spacer dots seems to influence the maximum size of a contact getting registered. Having a lower distance between spacer dots leads to only smaller objects shorting conductive layers. Larger objects' touches result in pressure being distributed across spacer dots leading to conductive layers not getting touched. This approach seems to be patented by Microsoft. Most touchscreen vendors use the average spacing of 3.5 mm and dot size around 0.1 mm [23].

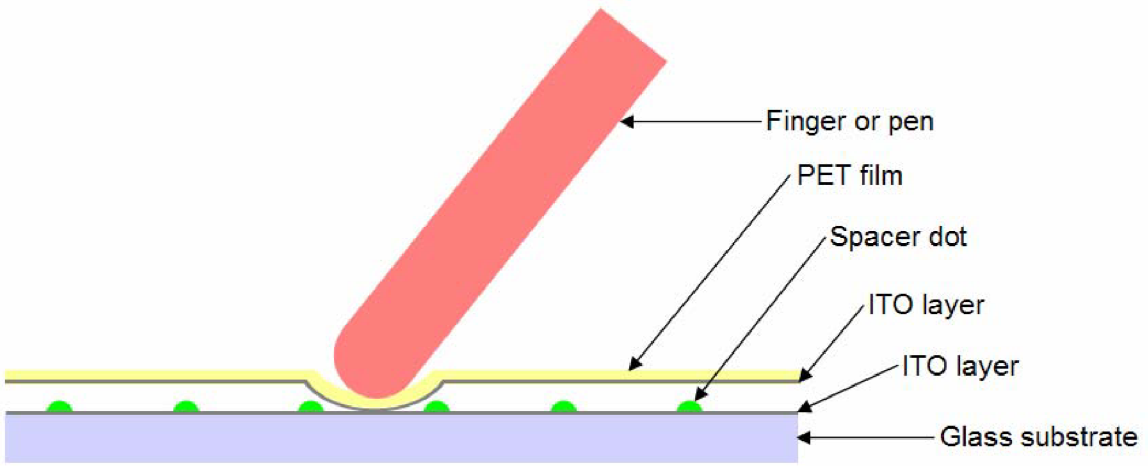


Figure 2.2: Components of touch screen [23]. ITO layers are conductive layers.

Palm rejection glove

The intention of this product is to cover all areas which should not be used for cursor control with a non-capacitive material. However, it never gained popularity and ended up being a gimmick.

2.2.2 Software solutions

The main advantage is that software solutions do not require any special hardware, and they are reasonably cross-platform. Applications rely on analyzing data from the touchpad. Extracting features like touch position, orientation, and size and using an algorithm to decide whether reject the input.

Another very different approach is a Hand Occlusion Model, showed in figure 2.3, proposed by Vogel et al. [22]. This model predicts from stylus location, where hand and forearm would be located, any inputs located in hand projected area are rejected, whereas outside area is accepted [22]. However, this approach appears to have low accuracy [2], though the model in this study was simplified. This model appears to be ineffective. Several hand positions lead to palm being outside wrist and forearm areas.

Annet et al. [2] investigated a comparison between several touch rejection algorithms. The algorithms explored were:

- A contact area algorithm, where input got rejected if four or more sensors on touchscreen were detecting touch.
- Pen hover algorithm, where if the pen was located at least 20 mm above the touchscreen, all inputs were rejected.
- Hand occlusion model combined hover algorithm. Static rejection region algorithm, shown in figure 2.4a, where depending if the user is left or right-handed, zones are projected, where area rejecting the screen is about 2/3 of the touchscreen.

For the right-hand user, the highest certainty is located on the right side of the touchpad, whereas on more left, input would get rejected. Stylus based rejection region algorithm 2.4b, similar to static rejection, but the boundaries move together with the stylus, together with

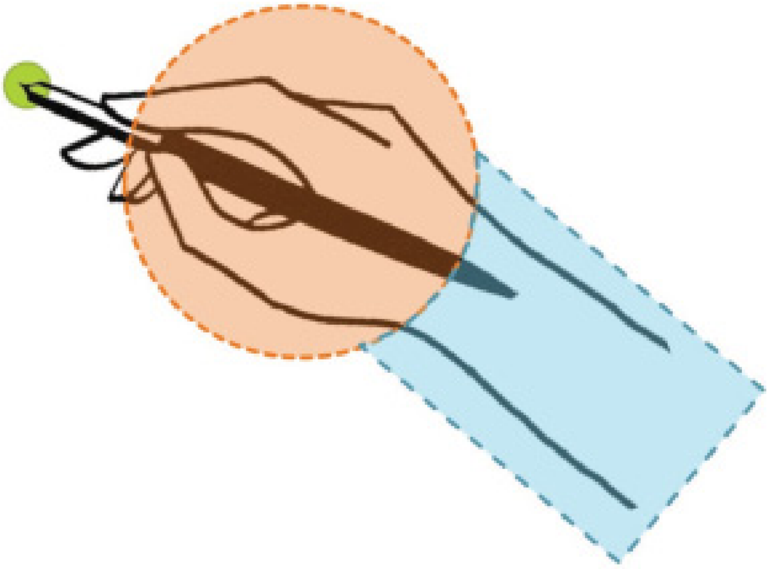


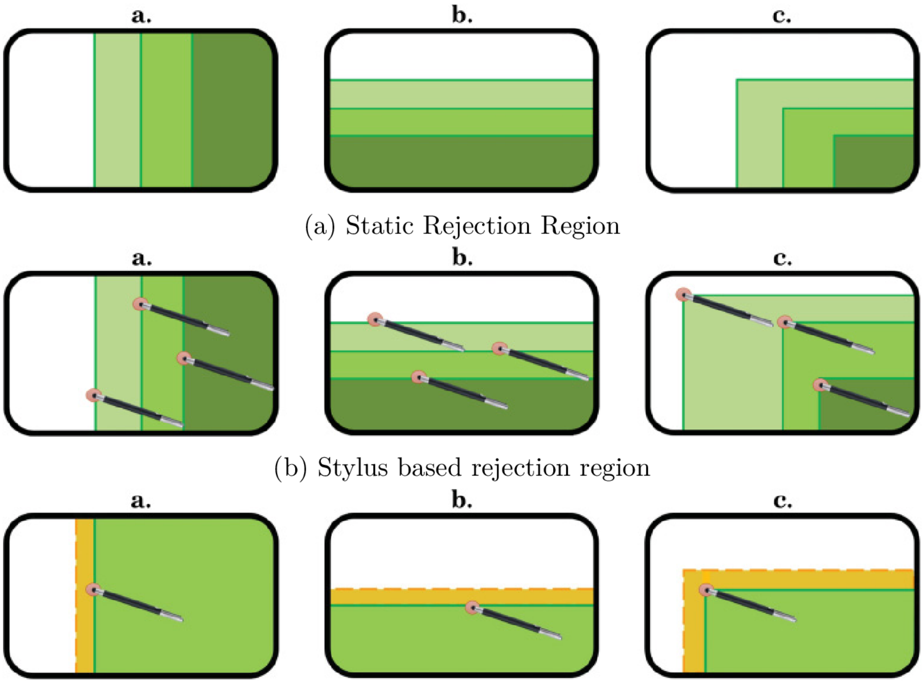
Figure 2.3: Vogel hand occlusion abstraction, adapted from [2]. Green area is location of a pen. Orange area is a predicted area of a hand. Blue area corresponds to abstracted forearm area.

a hover detection. Furthermore, for last, stylus-based rejection with a buffer, shown in figure 2.4c, buffer corresponds to an additional area around the stylus, where input would get still accepted. Hypothetical functionality is increasing hover height from 20 mm to 200 mm. The reason for it being hypothetical is that hardware in touchscreens cannot track pens above 20 mm, and instead, special additional hardware for tracking the pen was used.

All algorithms except contact area and static rejection region use a pen hover algorithm, which will not be possible for laptop touchpads because laptop touchpads do not use pens. Using only contact area does not produce great accuracy but could be used as a classifier. The hover algorithm is not usable for the reasons mentioned above. Vogel's hand occlusion model appeared to not be highly ineffective. Region rejection is probably not usable for laptop touchpads since it is likely users would use more entirety of the touch area than large touch screens. Usability for laptop touchpads, however, will need to be confirmed.

2.2.3 Software machine learning solutions

One of such already explored solutions is probabilistic palm rejection using spatiotemporal touch features and iterative classification. Application is not looking at current touch properties but their evolution over time [18]. This is useful because palm touches tend to have a long period of the same touches, and they do not change much over time. Pen or finger movements, however, move faster. Spatiotemporal touch features chosen for this model have five characteristics: the size of the touch area, sensors under palm touch area tend to flicker, palm points tend to be clustered close together, palms have only small movements [18]. Those features into a pre-trained decision tree, and classification is modeled. Similar features should also be experimented with within this work. This approach had a valid positive rate of 97.9 % and a false positive rate of 1.6 %, surpassing other publicly available palm rejection applications.



(a) Static Rejection Region

(b) Stylus based rejection region

(c) Stylus based rejection region with a buffer

Figure 2.4: Examples of region rejection models for right-handed users. For left-handed users, areas would be mirrored vertically. (a) vertical, (b) horizontal, (c) intersecting horizontally and vertically. Taken from Annett et al. [2].

There are rumors of Microsoft and Dell researching machine learning for palm rejection. However, this will need to be explored more.

2.3 Machine learning models

Machine learning can be defined as computational methods using learning to improve performance and make accurate predictions based on experience. Experience refers to past information available previously to learner, in form of learning, which is typically in form of electronic data. Such data are collected by interaction with the environment, we want the model to approximate. Most usually, data are in form of labelled dataset, which was human-labeled, as explained in a book Mohri et al. [14]. It is a branch of artificial intelligence. Learning algorithms can be applied to various applications such as text, language, speech, and vision classification. Beside classification, other applications are regression, ranking, clustering, dimensionality reduction and manifold learning [14]. Helping in various of areas, such as: game AI (such as „chess engines“), fraud detection, medical diagnosis, autonomous driving, biology etc.

In many areas, machine learning has already changed the shape of the many sub-industries. For example, in chess industry, AI models surpassed any human abilities and as the result, chess professionals train by exploring and mimicking strategies developed by AI, in order to stay on top of their ranking.

With current advances in computational technology, machine learning are getting more efficient, more complex, and adapted to entirely new problems. Currently largest machine

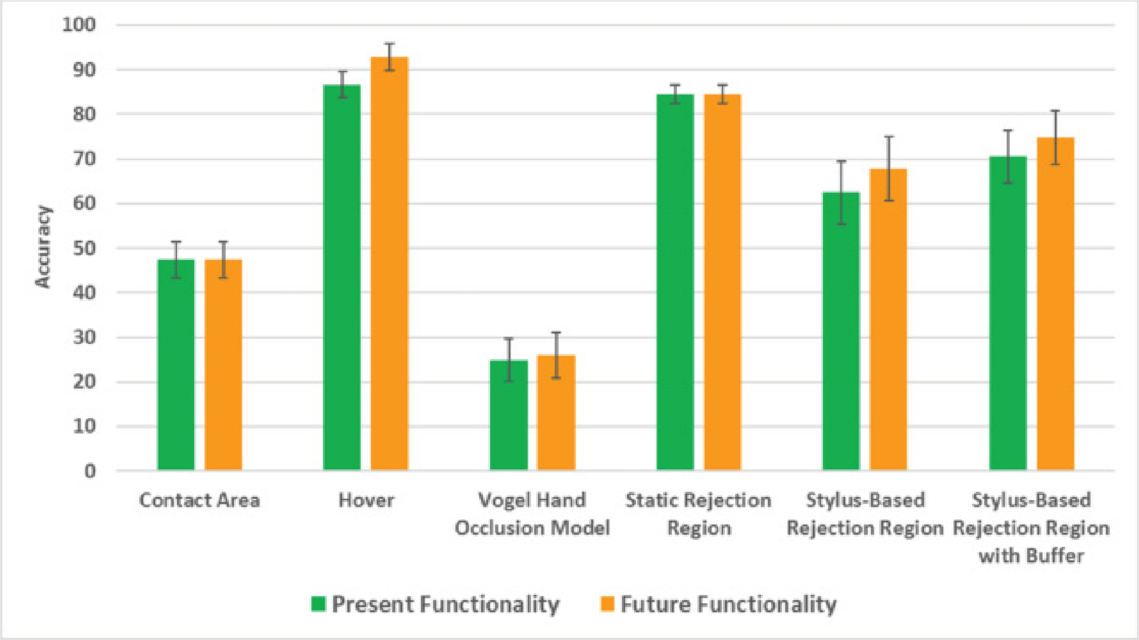


Figure 2.5: Comparison of unintended touch algorithms. Taken from from Annett et al.[2].

learning model is a deep learning auto regressive language model developed by OpenAI. Claimed to have up to 175 billion parameters. It's capability is still being explored.

2.3.1 Learning approaches

This subsection lists several learning approaches deployed in various machine learning models.

Supervised learning

By learning correctly labeled patterns, it receives some partial information about the true relationship between patterns and their labels. In the face recognition application, for example, a number of images are received, each labeled as either legitimate or fraudulent. In this manner, it is possible to learn to accurately label patterns from training data without wasting a great deal of design time and effort, and it can be applied to problems that are difficult to specify precisely in advance, perhaps because the environment is changing [4].

Training data set is a sample of input-output patterns. The result is a function that can yield desired output, given the input. The objects are already associated with target values before learning, thus the learning is supervised, it is told to the model how to interpret input data. The task is to find a deterministic function that maps any input to any output, predicting future input-output observations [14].

Unsupervised learning

Data is a sample of input patterns, without labels that would contain desired outputs. The unsupervised model uses the association to make a prediction of how much data correspond to the learned data set. It is often difficult to quantitatively evaluate a learner's performance in this situation due to the absence of labeled examples. In general, unsupervised learning refers to problems such as clustering and dimensionality reduction [14]. Unsupervised learning algorithms are generally more costly and less accurate than supervised learning models.

Reinforcement learning

Instead of passively receiving labeled data in supervised learning, the learner is actively involved in the process. Through interaction with the environment, reinforcement learning gathers information. Following an action, the learner receives two types of information: the current state of the environment, and a reward, which relates to the corresponding goal.

Deep learning

Deep learning is a machine learning method that uses artificial neural networks. The main difference in this type of learning is that feature extraction is learned from data, features typically are not fed into the model. This is useful in cases where the best features are not known.

Shallow learning

This is an opposite to deep learning. Shallow learning models are fed with features, and are not capable of feature extraction.

2.3.2 Gaussian Mixture Model

This subsection is adapted from Reynolds [16].

A Gaussian Mixture Model (GMM) is a parametric probability density function that is represented as a weighted sum of Gaussian component densities. These models are commonly used to calculate probability distributions for continuous measurements or features in biometric systems, including spectral information related to vocal tracts in speaker recognition systems. In GMM, parameters are estimated from training data using either the iterative Expectation-Maximization (EM) algorithm or the Maximum A Posteriori (MAP) estimation.

The Gaussian mixture model is an unsupervised learning model. The Gaussian mixture is a function that is comprised of several Gaussians. Each Gaussian is described by a gaussian function. Gaussian mixture model works best with data having Gaussian distribution's nature. Gaussian distribution is observed to be occurring in nature. Since most touches tend to have a circular shape with pressure being higher in the center, using a uni-variate Gaussian mixture could be possible.

A Gaussian mixture model is a weighted sum of M component Gaussian densities as given by the equation,

$$p(x|\lambda) = \sum_{i=1}^{M} w_i \, g(x|\mu_i, \Sigma_i) \tag{2.1}$$

where,

- x is a D-dimensional continuous-valued data vector (i.e. measurement or features),
- w_i, $i = 1, ..., M$, are the mixture weights,

- and $g(x|\mu_i, \Sigma_i)$, $i = 1, ..., M$, are the component Gaussian densities.

Each component density is a D-variate Gaussian function of the form,

$$g(x|\mu_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^D \, |\Sigma_i|}} \exp^{-\frac{1}{2}(x-\mu_i)\, \Sigma_i^{-1}\, (x-\mu_i)} \tag{2.2}$$

with mean vector μ_i and covariance matrix Σ_i. The mixture weights satisfy the constraint that $\sum_{i=1}^{M} w_i = 1$. The complete Gaussian mixture model is parameterized by the mean vectors, covariance matrices and mixture weights from all component densities. These parameters are collectively represented by the notation,

$$\lambda = \{w_i, \mu_i, \Sigma_i\} \quad i = 1, ..., M. \tag{2.3}$$

Due to their ability to represent a wide variety of sample distributions, GMMs are frequently used in biometric systems, most notably in speaker recognition systems. Its ability to approximate arbitrarily shaped densities smoothly is one of the GMM's strongest assets.

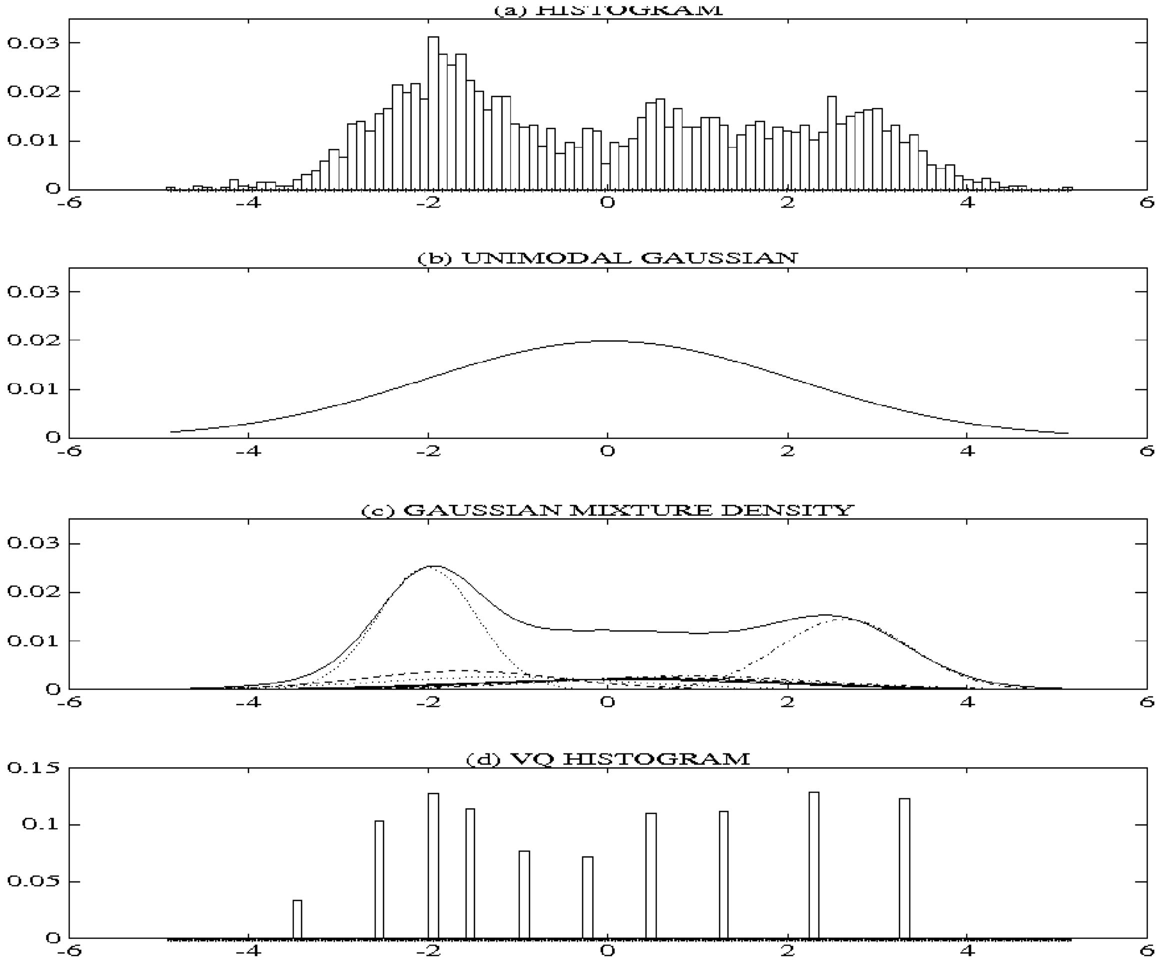


Figure 2.6: From Reynolds [16]. Comparison of distribution modeling. (a) histogram of a single cepstral coefficient from a 25 second utterance by a male speaker (b) maximum likelihood uni-modal Gaussian model (c) GMM and its 10 underlying component densities (d) histogram of the data assigned to the VQ centroid locations of a 10 element codebook.

2.3.3 Linear regression

Linear regression is a statistical approach for modeling the relationship between a predictor and an independent variable. When a relation is deterministic, one variable can be express by the other. For example, using a variable for distance in miles, the second variable for kilometers can be accurately predicted by the first one. The idea is to find a line that best fits all data.

2.3.4 Logistic regression

Logistic regression has very similar steps to linear regression, except it uses a binomial response variable instead of a continuous one. It can be seen in both the choice of parametric model and the assumptions that are used in logistic regression and linear regression. Following the same principles used in linear regression, when this difference is taken into account, the methods used in a logistic regression analysis are identical.

Binary response data are most often modeled with logistic regression. Typically, binary responses take the form of 1/0, with 1 generally indicating a success and 0 a failure. There are numerous ways in which 1 and 0 can be defined, depending on the nature of the study. As a general rule, response 1 indicates the foremost subject of interest for which a binary response study is designed. Using normal linear regression to model a binary response variable would introduce substantial bias into the parameter estimates. Standard linear models assume that the response and error terms have normal or Gaussian distributions, observations in the model are independent, and the variance, σ^2, is constant across observations. When a binary variable is modeled using this method, distribution and variance assumptions are violated [11].

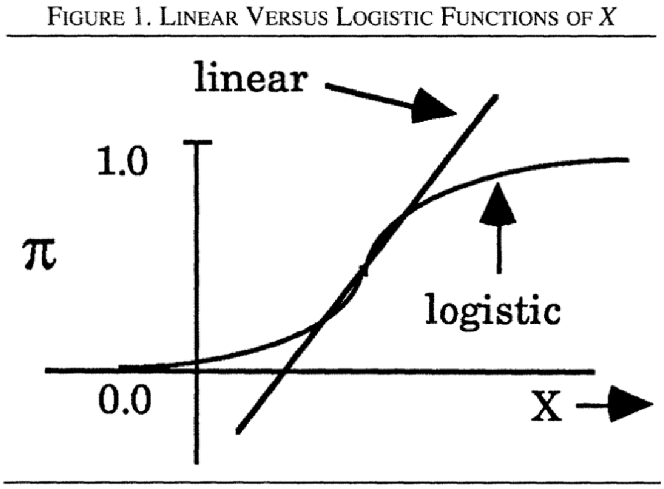


Figure 2.7: Linear regression vs Logistic regression. Taken from [8].

2.3.5 Support Vector Machine

This subsection is adapted from Tripathi [21].

A support vector machine is a type of discriminative classifier that is formally defined by a separating decision function. An SVM decision function can be viewed as an optimal hyperplane that serves to identify observations based on patterns of information about those observations, referred to as „features.“. A hyperplane can then be used to determine the most probable label for not yet seen data. The features used for inferring the hyperplane are rarely raw data; instead, they are derivatives resulting from some form of interpolation. Support vectors are derived from the relationships between features and are referenced by coordinates. As a result, the algorithm outputs an optimal hyperplane that categorizes new examples when given labeled training data (supervised learning) with the maximum distance between both classes' data points.. Two-dimensionally, this hyperplane can be characterized as a line cutting a plane in half, with each class lying on one side. It can be also used for regression analysis, but classification is mostly used. It is considered as very effective classification algorithm. In regular variant support, vector machine algorithms can separate only two non-overlapping classes. The result of the support vector machine is not represented as the probability of representing a class. Instead, it produces a score.

The SVM algorithm uses a set of mathematical functions called Kernels. In some classification problems, it is not possible to find a hyperplane or a linear decision boundary (figure 2.10). We may obtain a hyperplane in the projected dimension if we project the data into a higher dimensional space compared to the original space. In this way, Kernel aids the search for hyperplanes in higher dimensional spaces without increasing computational cost. With an increase in dimensions, the computational cost typically increases.

SVM optimization uses a regularization parameter (called `C` in Python's sklearn library) to determine how much it wants to avoid misclassifying the training examples when optimizing the model. Optimization will choose a smaller-margin hyperplane for large values of C if that hyperplane does a better job of classifying all the training points correctly. On the other hand, a very small value of `C` will cause the optimizer to look for a larger-margin separating hyperplane, even if it misclassified more points.

Gamma describes how far the influence of a single training example extends, with a low value meaning 'far' and a high value meaning 'close'. Alternatively, when gamma is low, the calculation for the separation line takes into account points that are away from plausible separation lines. In contrast, high gamma means that the points close to plausible lines are considered.

Also, some hyperparameters are specific to Classification or Regression Problems, or they are used along with any specific and dependent hyperparameter.

SVM has a high time complexity $O(n3)$ for LibSVM implementation [1] and as the result, it's demanding, and it is not recommended for learning datasets with size $>$ 100,000. SVM can be capable of either linear or nonlinear classification (figure 2.10), but usually it's the former. Linear SVM has advantage of having considerably lower time complexity. The use of SVM is widespread in many areas, such as disease detection, text categorization, software defect, intruder detection, time-series forecasting, detection, etc.

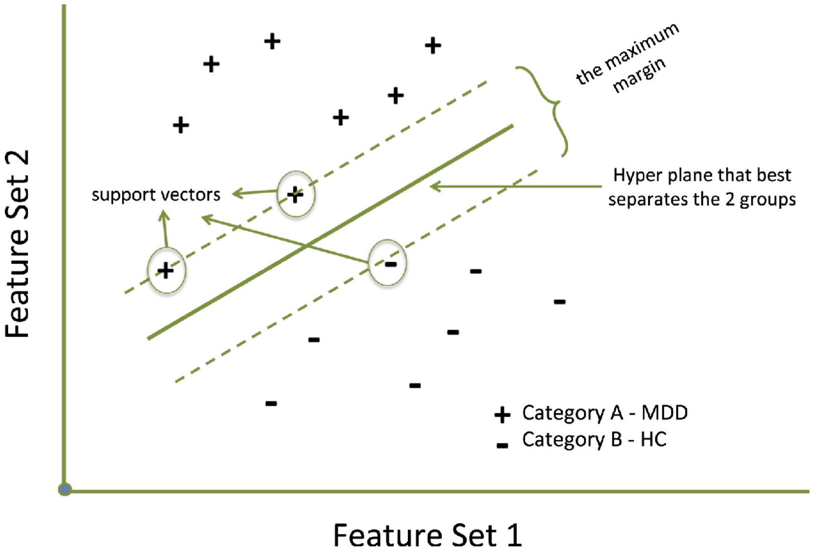


Figure 2.8: Illustration of the hyperplane that separates the support vectors. Taken from [17].

2.3.6 Random Forest

Random forests are a scheme proposed by Leo Breiman in the 2000's for building a predictor ensemble with a set of decision trees that frow in randomly selected subspaces of data. It's strength were analysed in the next 10 years, where they were proven to be very strong, and one of the most accurate general-purpose learning techniques available. Random forests ideas are influenced by the early work on geometric feature selections, random subspace methods and random split selection approaches [5].

Leo Breiman in [6] explains random forests as a combination of tree predictors in which each tree is influenced by a vector of values selected at random and with the same distribution throughout the forest. As the number of trees for forests increases, the generalization error almost certainly reaches a limit. The generalization error of a forest of tree classifiers depends on the strength of the trees within the forest and their correlation. Internal estimates are used to monitor error, strength, and correlation, and these are used to measure the effect of increasing the number of features used in the splitting. Variable importance is also measured using internal estimates. Regression is also applicable with these ideas.

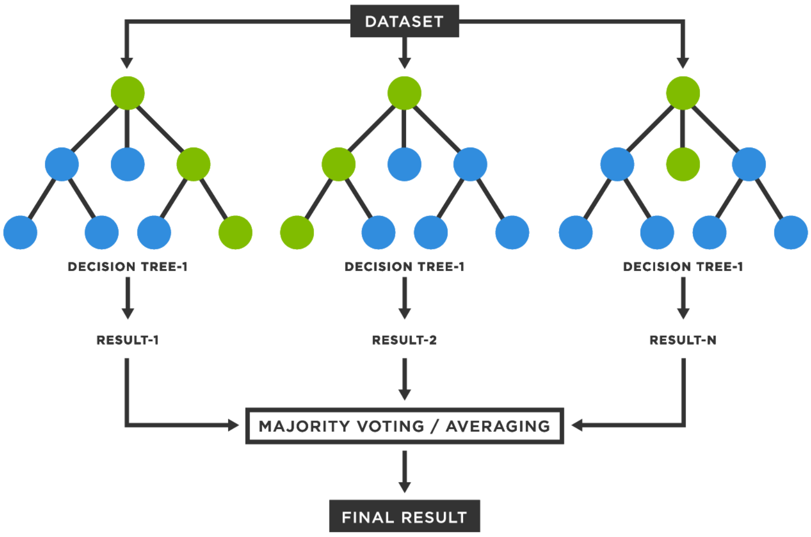


Figure 2.9: Random forests scheme taken from [3]

As a random forest model, hyperparameters can be either used to increase the model's predictive ability or make it run faster. The first hyperparameter is `n_estimators`, which determines how many trees are built before a maximum vote is taken or before averages of predictions are calculated. There is generally an increase in performance and stability with more trees, but the computation becomes slower with more trees. The `max_features` hyperparameter determines how many features are used to split a node. The last hyperparameter is texttmin_sample_leaf, which determines how many leaves are needed to split an internal node. It has some advantages for people who have not yet been exposed to machine learning and coding since it is very versatile, requires very little feature analysis compared to other types of models, and even with the default hyperparameters, is very accurate with little need to change them. A key issue in machine learning is overfitting, but random forests do not suffer from this problem. Furthermore, although random forests are fast to train as long as the number of trees is not extremely high, the calculation of the prediction is not that fast [9].

2.4 Artificial neural networks

Artificial neural networks (ANNs) are the foundation of deep learning. In the last decade, artificial neural networks have become popular for applications ranging from financial prediction to machine vision. These networks were originally designed to be simplified versions of biological neural networks. However, their focus has shifted from biological neural networks to supervised learning problems. We neglect the word 'artificial,' and we consider neural networks as nothing more than a type of nonlinear function (figure 2.10).

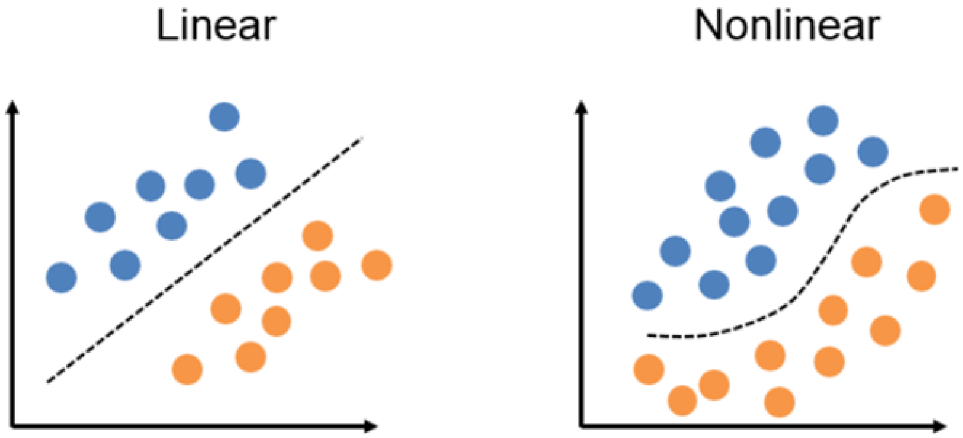


Figure 2.10: Linear vs. nonlinear problems. Source Sebastian Raschka [15]

Neural networks were named after neurons in biology and connections between them. In biology, a neuron is a type of cell found in the brain that accepts input and responds. Inputs and outputs are in the form of electrical signals. However, artificial neural networks are different from neural networks within a brain. Especially, learning methods of artificial neural networks are not inspired by human processes. A biological neural network's learning mechanism is unknown.

Before the term artificial neural network was introduced, models that fit the description of neural networks existed. A popular model that was created before ANNs were introduced is the Multi-layer perceptron (MLP), but mathematical interpretations of neurons existed even before this. Mathematical model of a perceptron is represented in equation 2.4. Nowadays, it's classified as an ANN. Due to more accurate options available nowadays, MLPs are rarely used in practice, but they are used to teach deep learning basics.

If a neuron network is large enough, it can model any function [12][13]. The process of neural network learning is referred to as deep learning. In deep learning, neurons' activation functions change when misclassification occurs.

A typical artificial neuron network consists of three types of layers. An input layer, an output layer, and hidden layers. The hidden layer's cells are hidden from everyone, including the network's developer, and are adjusted only through learning methods.

Convolutional neural networks introduce a new type of layer into neural network structure, a convolutional layer. Convolution is a process of combining two informations into a new information, typically by applying a filter. It is inspired by biological processes in brain, resembling a visual cortex. It is used mainly for deep learning problems with image data.

Recurrent neural networks introduce memory capabilities to neural networks. By using memory processes, they are able to process sequence of inputs and extract contexts.

2.4.1 Perceptron

Perceptron calculates a function from $\mathbb{R}^n$ to $0, 1$. Network with of type of neuron, having an output of either 0 or 1 can be used for pattern classification problems where we want to divide patterns into two classes, labelled '0' and '1'. A perceptron computes a function f of the form:

$$f(x) = sgn(w \cdot x - \theta), \tag{2.4}$$

for input vector $x \in \mathbb{R}$ where:

- $w = (w_1, ..., w_n) \in \mathbb{R}$ is an input weight for each precedent neuron connected, and
- $\theta \in \mathbb{R}$ is a bias (threshold weight).

$w \cdot x$ denotes the inner product $\sum_{i=0}^{n} w_i x_i$, and

$$sgn(\alpha) = \begin{cases} 1 & \text{if } \alpha \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

When learning, on misclassification, weights get update to move the decision boundary towards a misclassified example.

2.4.2 Cost Function

Cost function is used to determine a relation between weights and biases and correct output of the network. To find the best weights configuration, a minimum of cost function can be found. However, finding minimum in multidimensional space is an intricate problem. Cost function also can have multiple local minimums, and finding global minimum is a difficult task. Usually because of how complex cost functions are, only local minimums are found by approximation.

2.4.3 Gradient Descent

Gradient descent is a method for finding a minimum of a function. Gradient descent cannot find always the best result. Gradient descent can be abstracted as dropping a ball at a 3 dimensional space. When ball touches any surface, it rolls down according to angle of the surface hit. In a same way as gradient descent, dropping a ball, doesn't have to find the best result, when there are more local minimums. Simulating ball physics isn't needed to find such minimum, this is only an oversimplified explanation. The most elementary gradient descent has a fixed step, and when a „ball“ is crossed with a function, the new trajectory by a function slope, which can be calculated as derivatives of the function at certain position. More advanced back-propagation methods usually optimize this process.

2.4.4 Back-propagation

Back-propagation is a learning algorithm for training neural networks. It is generally faster than calculating a gradient descent of a cost function. It works by propagating desired output in the direction of the input (backwards), on each neuron checking all input neurons in previous layer and changing the weights to better match the desired function. And in the same way, adjusting the next layer and so on.

Perceptron is one of the earliest machine learning algorithms. It is a binary linear 2.10 classification algorithm. It learns a decision function by processing training points one at a time. A binary function is a function whether input belongs to some specific class. It is a linear classifier. If data cannot be linearly separable, it does not converge all points to the correct class. Perceptron is an artificial neuron with an activation function. A network of perceptrons could be technically classified as a simple artificial neural network. The output from the neural network used for classification is a probability of data being a specific class [4].

A function of perceptron is as follows:

$f(x) = sgn(wx - \theta)$

where w and θ are adjustable parameters. Output of the function is $0, 1$.

Decision boundary corresponds to $wx - \theta = 0$.

When learning, the perceptron algorithm starts with arbitrary values and when it misclassifies data, parameters get updated: $w = w + \eta(y - f(x))x$, where η is a prescribed fixed positive constant. The result of this update is decision boundary moving closer towards misclassified point x.

Anomaly detection

Analyzing data for anomalies involves finding patterns that differ from expected behavior. The terms anomaly and outlier are often used interchangeably in the context of anomaly detection. As an example, anomaly detection is extensively used in the detection of fraud for credit cards, health care services, cyber-security intrusions, faults in critical safety systems, and military spying.

Chapter 3

Implementation

This chapter discusses the methods used in this work and ways to approach it, such as tools for creating the dataset, tools to gather more data for deep learning, walking through several prototypes of machine learning, and implementing a final program that uses one chosen model and demonstrates a model in real-time.

3.1 Data collection

This section will talk about the implementation of algorithms, that made it possible to get a dataset for learning algorithms. It is common for capturing sessions to contain many data frames, where there were no touches at the time, so a model could be created to locate and eliminate these images. Due to low amount of training data, it might be necessary to manipulate the data to increase the accuracy of deep learning models so that enough data is available, thus several data augmentation algorithms are proposed and implemented.

3.1.1 Raw data capture

The only way to capture raw data is to enable debug mode on the RMI touchpad driver. During driver compilation, it is necessary to specify `CONFIG/_RMI4/_F54=y` when compiling drivers from Linux repository. It is then necessary to load the driver with `debug/_flags=1` as an `insmod` argument command.

The driver passes raw data through a file stream, usually located in `/dev/v4l-touch0`. It is likely that touchpad performance will be greatly compromised as a result of the debug mode's excessive amounts of data traffic. Because of this, only laptop manufactures are able to properly adapt palm rejection. In the best-case scenario, this will encourage them to develop hardware that would allow palm rejection implemented outside firmware to be efficient and lag-free, enabling better public researches about this problem.

Because of the limitations with the hardware for collecting data, explained in section 2.1, data had to be collected only by me. Because this project aims to experiment more with deep learning, the project requires a good amount of training data. As a result, only selected data collection approaches are practical from a time standpoint.

As soon as this step was completed, the next step was figuring out how to obtain any decent amount of data, while having them properly labeled. With the first attempt of collecting data, an approach of manually positioning a hand and then capturing this moment as one image was used. Trying to use this method quickly proved to be very time-consuming, as well as creating very unnatural results, and so another approach was needed.

The second method involves utilizing the computer's system in order to obtain information about the rejection. It could be done, for instance, by tracking cursor movements in the system and labeling data accordingly. The approach nonetheless seemed complicated, and, first of all, the touch rejection on the ThinkPad T580 used is, based on my observations, inadequate, where the only case when it rejects touches are if the touch is on the left edge of the touchpad. This would result in a dataset that wasn't labeled accurately enough, severely limiting the model's potential.

The last method focuses on having label-specific scenarios, wherein for each data collection scenario, only selected types of touches, matching desired training classes, are collected. In spite of the reduction in mislabelling risk, the results may not be as a great representation of real-world scenarios as the previous method. For a dataset collection containing legal data (data that should be accepted), the touchpad will not be touched with the palm, for example, in sessions involving playing a cursor-only game. Likewise, to record palm data, having wrists rested on the laptop while writing some long text, as well as experimenting with hand resting and typing positions. This third approach was chosen for creating training and test datasets. However, this approach also relies on removing anomalous data from the dataset containing illegal data.

3.1.2 Data elimination

Data collection faces multiple problems, explained in previous subsection 3.1.1, that determined how data were captured. Due to this, the dataset contains scenarios when the touchpad is not touched, or it may happen just after the touchpad was touched. It was, therefore, necessary to establish a way of identifying insufficient data that could confuse learning models.

In order to remove unsatisfactory data, a model must be developed. Normally, this type of analysis falls under anomaly detection; however, anomaly detection tends to deal with anomalies that are sparsely represented in the dataset, thus requiring statistical approaches. However in this situation, unsatisfactory data are more substantially contained in the dataset, because of that, statistic methods are not explored and the shallow learning approach is explored.

This problem involved creating two classes of data. The first class contains all touch values from the palm rejection training dataset, both legal (fingers) and illegal (palms). The second dataset was manually created and aimed to contain two specific scenarios. The first situation is when the user isn't touching the touchpad, and the images consist mainly of blank images. TThe second situation aimed to capture images that happen right after the user stopped touching the touchpad; the reason is that this situation produced unreliable data, and I wanted the model to learn not to reject this situation. Those inconsistent data, after a while, get into an equilibrium that corresponds to the first situation. This effect resembled heat residue after touching a cold surface and viewing it with a thermal camera, but the residual values reached negative numbers. This second dataset consists of only 265 images; however, since very few scenarios were needed to cover this dataset, this size was sufficient.

Data were analyzed for valuable features. Features explored were:

- Minimum value (min)

- Maximum value (max)
- Mean
- Variance (var)
- Sum
- Peak to peak value (ptp)
- Standard deviation (std)
- A sum across diagonals (trace)

Feature selection was performed on the newly established dataset. Three features were distinct when plotting their densities for each class: max, ptp, and standard. Although these visualizations are helpful for selecting features, a more objective method was needed. With a Random Forest classifier, the importance of each feature was extracted and plotted, where it's clear which features are useful for classification. As some features determine very similar factors, using them together would not enhance the classifier's accuracy, but quite the opposite. It is thus vital when selecting features to consider the correlation between them. This is done with a correlation plot, which can be seen on figure 3.2. A correlation of above 0.9 (or below -0.9) is usually considered too high, and those features shouldn't be used together. Correlation values of 1 (or -1) indicate a linear relationship, which means the features are often identical but interpreted differently. I chose variance as the first feature, due to high importance, and as the second feature maximum value, due to low correlation between them. Although standard deviation would be a better alternative, due to the lower correlation with maximum value, the original option is accurate enough for this problem.

The „target" column is not considered a feature, but rather a label on the data that indicates if it is legal or illegal (finger type, palm type). The feature for max value has a very low importance value. However, all values with higher importances were too correlated with a variance feature.

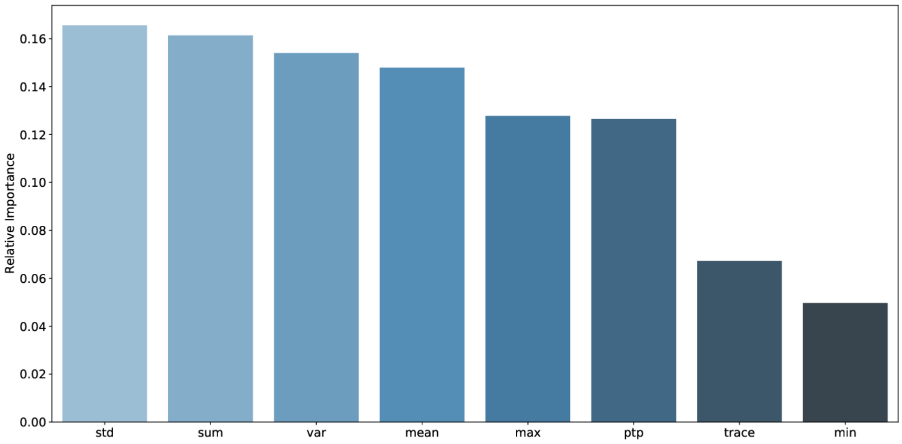


Figure 3.1: Feature importances according to Random Forest algorithm

Since Gaussian Mixture Models don't generate decision values and generate likelihoods instead, a decision threshold is used to label data based on the model's predictions. A

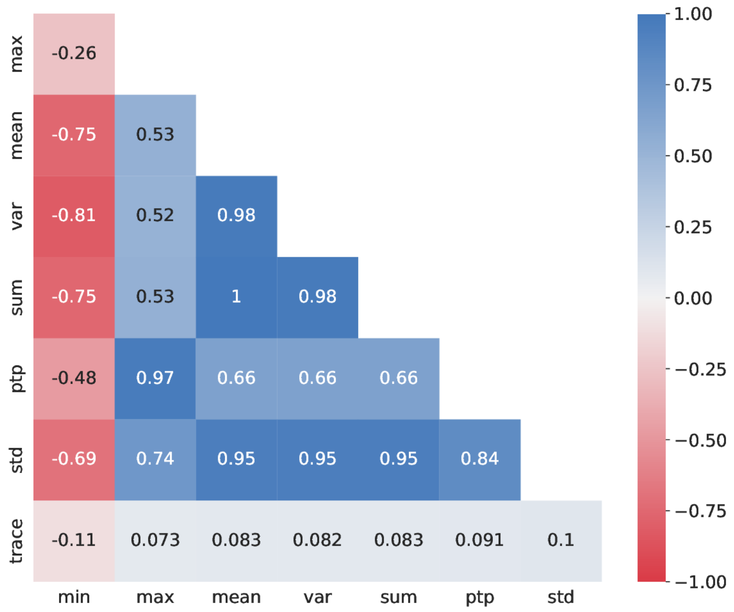


Figure 3.2: Correlation of features

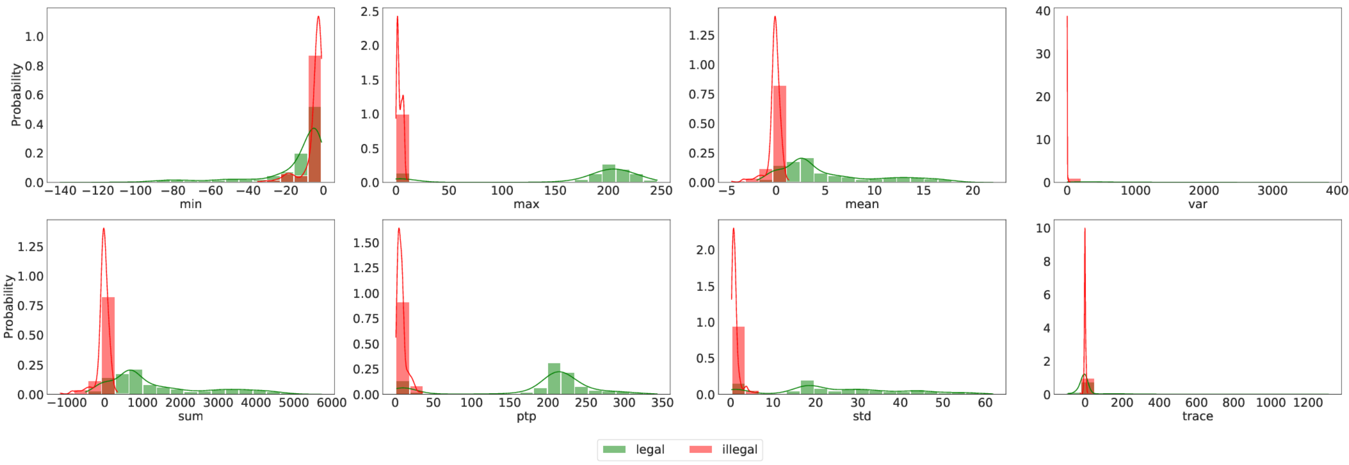


Figure 3.3: Probability plots of each feature

logarithmic likelihood of -500 proved to be a good threshold for this decision. The likelihood of data exceeding this value is considered „non-touch“ and for illegal datasets, they are eliminated, as there is no reason to reject data where no touch is detected.

3.1.3 Data augmentation

Models that use deep learning require a lot of data. In addition, we cannot control what deep learning models will focus on. For example, deep learning models could learn to reject touches on the right side of the touchpad based on touch position. Among the ways to mitigate these problems is to train your model with very diverse touch positions. Three techniques are used to modify the data in this thesis to increase the dataset size: mirroring (flipping), shifting, and rotating.

Mirroring

It can be used to simulate an other-handed user and can be applied to any image without limitations. Using mirroring is primarily useful for illegal palm data, where touches aren't uniform and the image can't be shifted much. To implement mirroring, the function `numpy.flip` is used, and it was the simplest of the three data augmentation solutions explored. This data augmentation approach doubles data.

Numpy is a Python library for n-dimensional arrays. It also provides wide range of operations for working with such arrays. N-dimensional arrays can be created in Python, but working with them is not as easy, as well as Numpy can be a lot of faster.

Shifting

The method shifts touches within the image's range. A range of free motion is computed across each image. When shifting data, the first problem is to decide how much to shift so that data still retain some of its nature and do not lose any detail. Therefore, the indices of all cells containing values above 25 are extracted. These indices are used to draw a rectangle around the cells, this rectangle is shown in figure 3.4. In order to not affect the features of the touches, this rectangle was decided not to be able to exit outwards from the image's frame. In addition, if the rectangle touches any edge already, it must keep touching that edge; if it touches both axes, the image cannot be shifted. The range of motion the image can be shifted depends on these rules. A median of the entire image fills in the missing values caused by shifting since large touches wouldn't allow the image to move in any case. Using this augmentation method, it was possible to produce a dataset that was 100 times larger than the original.

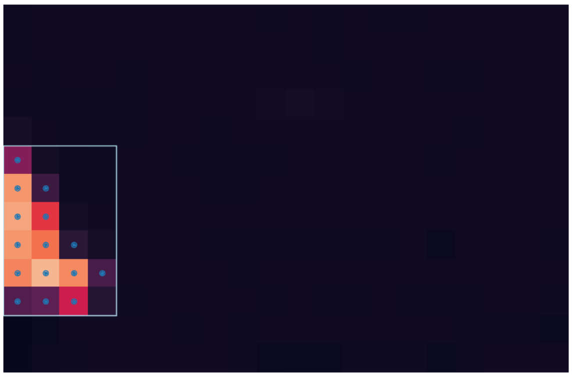


Figure 3.4: Visualisation of the „rectangle“. Where dots mean a significant pixel

A pixel-by-pixel shift is realized. Shifting by less than one pixel is also possible, but this results in more distortions, such as stretching and losing detail (smoothing). Additionally, the image will have higher values than the original one as shown on figure 3.5.

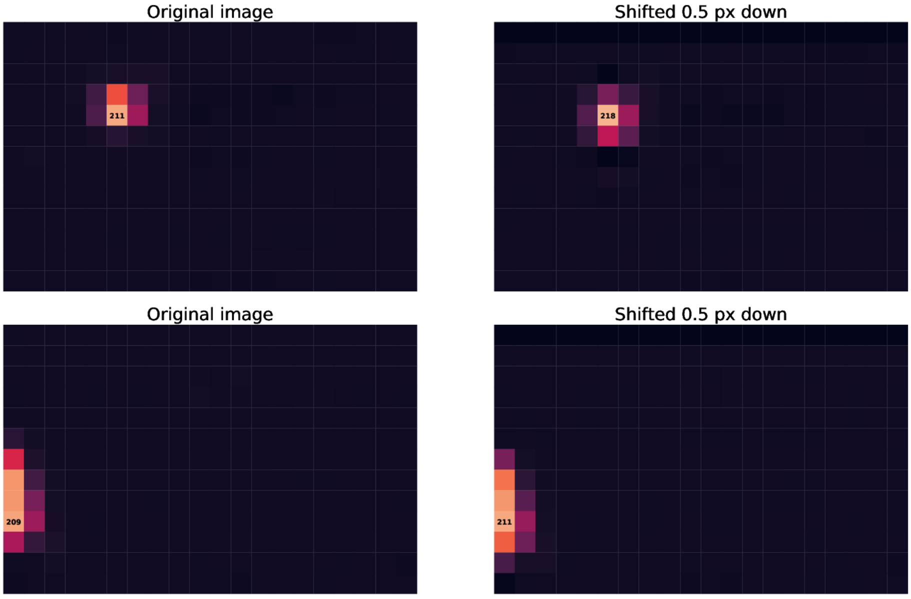


Figure 3.5: Shifting per half pixel.

Since palm data have larger touch areas and those touch areas usually touch edges of the image. This method cannot generate as much illegal data as legal, which changes the ratio between legal and illegal touches. Shifting is performed by a `scipy.ndimage.shift` function.

Rotating

Among the three explored methods, rotation of the touches is the most complicated. Additionally, it usually plays a more significant role in the creation of illegal data, which helps rectify ratio imbalances caused by shifting limitations. Also, it may be beneficial for diversifying legal data containing multiple touches at once.

A function called „scipy.ndimage.rotate“ is used to rotate data. The main advantage of this function is that it covers the biggest obstacle in rotation: Filling in the gap created by rotated space beyond image boundaries into the space in boundaries, resulting in blank areas without values if rotation is inwards toward the center. Various modes were available for overcoming this, but satisfactory results were only obtained with three: the „reflect“, „nearest“ and „mirror“ modes, from whose 'nearest' was chosen.

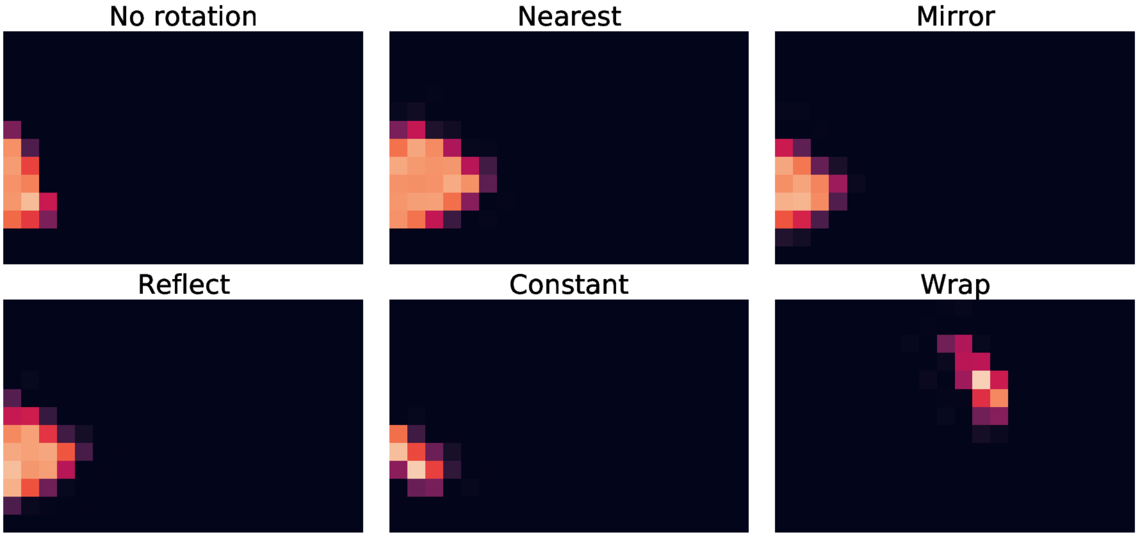


Figure 3.6: Rotation mode of filling outside boundaries.

A strong limitation exists despite the benefits of this function. The image can only be rotated with the center as the reference point. This caused touches to move across the image, and even out of the image's frame. Rotating images with the center of the touch as a reference point fixes those problems.

To overcome this, a way to find a center of the touch has to be found. Because of time limitations, the center is only approximated. The approximation is made by calculating an average index weighted by a sum of the entire column or row. Essentially, a sum is calculated across columns and rows, then, each value in this sum is summed and multiplied with its index. This index corresponds to a coordinate of each column or row. The result of this sum is an X or Y coordinate for an approximation of the center touch. For this to work, all cells that are not part of touch itself have to be zero. Due to this, all values below 35 are replaced with a zero value.

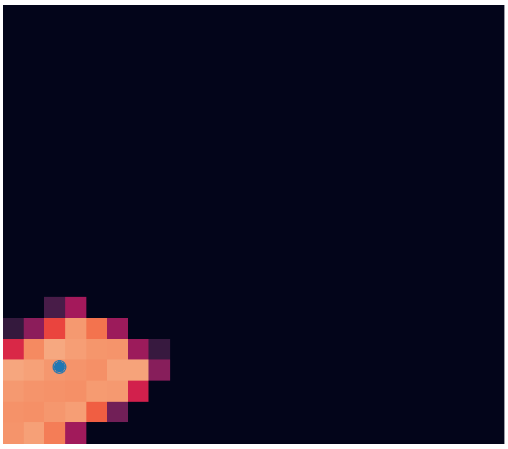


Figure 3.7: Image's heat-map with calculated center using proposed algorithm shown as blue dot.

When using „scipy.ndimage.rotate", the image's dimensions get larger after the rotation, since the image's frame uses the least amount of space horizontally and a new image frame is adjusted to cover all rotated cells, the frame rotates with the image. This rotation function provides the option of cropping the result rotated image in order to maintain original dimensions. This cropping, on the other hand, occurs at the center of the image, causing touches to shift, and with high rotation angles even moving out of the image. By using the touch center algorithm previously created for rotation, the image is shifted so that the new center after rotation moves to the original image's center, and it is cropped by array splicing. Shifting can be again done only per pixel, so the image isn't shifted to the same exact point.

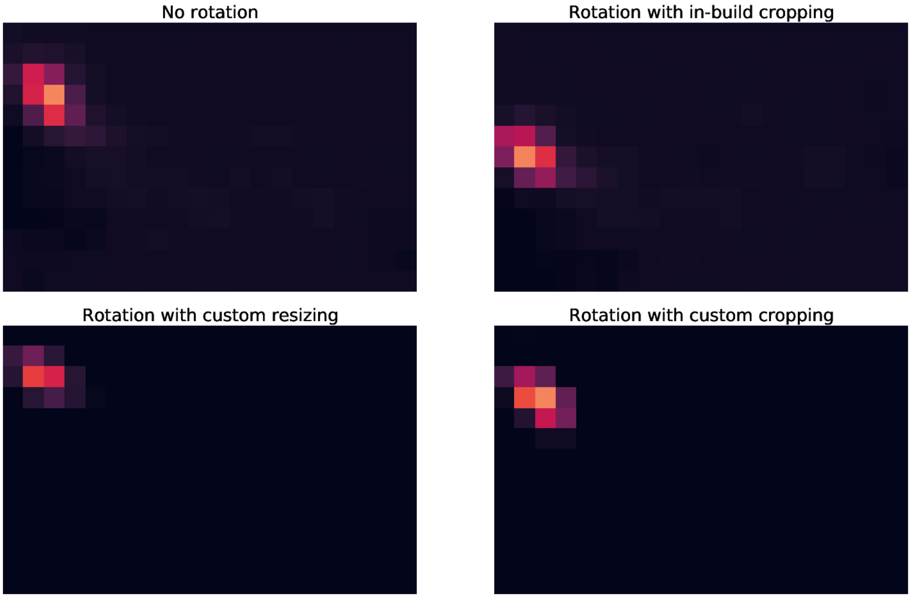


Figure 3.8: Resize techniques used for rotation

This works reliability enough for rotation to not get shifted in any significant way. However, if there are multiple touches, the center is usually between those touches. This approach was chosen because of its low time complexity. A possible alternative was using k-means clustering.

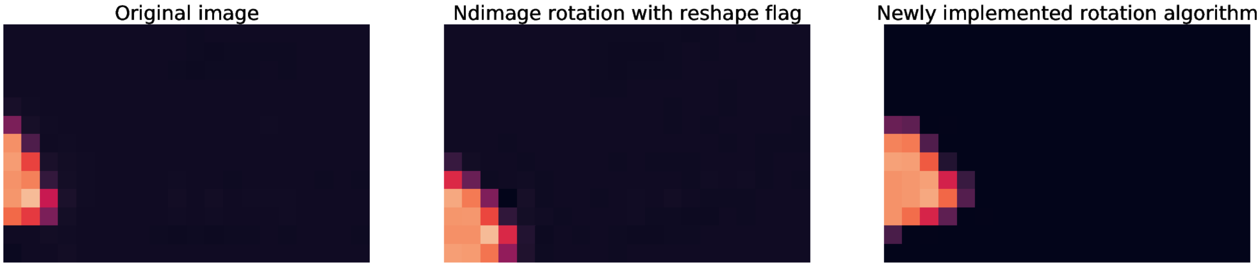


Figure 3.9: Comparison between the old and new rotation algorithm.

3.2 AI/ML prototyping

This chapter discusses models which were experimented with in order to see how they behave for this problem, and how are they usable in achieving solution to the problem this thesis is about.

3.2.1 Deep learning models

This chapter explains procedures that were taken to create prototypes of basic deep learning models. In order to understand their behaviour with touchpad dataset, how well they can be applied for this thesis' problems and how well they compare to shallow learning models. Models explored are: Artificial Neural Network (ANN), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Focus is on general overview, not on analysing complex behaviours of learned models.

Deep learning models are implemented using a Keras library. Which is a high-level Python interface for TensorFlow library. However, it also supports other deep learning back-ends, such as: Theano, PlaidML, MXNet and CNTK (Microsoft cognitive toolkit). Purpose of Keras is mainly prototyping artificial neural networks and learning basics about deep learning without needing to know how to work with tensors and other such concepts.

Training data

Unlike shallow learning models, deep learning models are additionally fed with augmented data in order to meet their high quantity dataset requirements. This is however a problem for recurrent neural networks, since they require sequential data with context, shifting, at least how it is implemented here, destroys this context. As the result, RNN in overall was fed with less data than ANN and CNN. However, for comparison purposes, models were also compared between each other with data gathered for RNN.

Datasets for ANN and CNN are augmented; shifted, mirrored and rotated. For rotation, all possible shifting in range of motion is used, but rotation has more options, and only some rotation can be done. Rotation is mainly done for illegal data, to balance out the ratio between legal and illegal data, where legal data were augmented more because of free-er range of motion.

As range of rotation. Only integer angles are used. To use rotation for the generation of data, a range of motion has to be defined. Where above this range of motion, generated data would not be corresponding to real data. The first idea was an analytical approach. From a dataset containing real data, angles of each touch would be pulled, for example by fitting an oval over the touch area and calculating the rotation of such oval. This could be then fitted on a normal distribution and used this distribution for determining angle for rotation to achieve a generation of new data. However, this was not explored to save time.

Artificial Neural Network

ANN networks implemented, unlike of the other two options, CNN and RNN, consists only of artificial neurons.

This network achieved a decent accuracy even with only 2 neurons in hidden network. Showing that this problem is not very complex for a neural network.

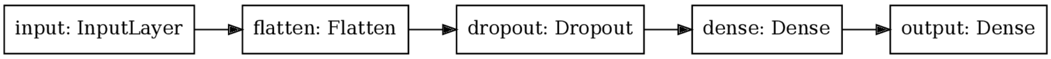


Figure 3.10: A general graph of Artificial Neural Network used.

Convolutional Neural Network

For convolution, another layer in hidden layer is added, a convolutional one. Usually, convolutional neural networks are applied to high resolution images. And there a polling layer is needed after the convolutional layer. But here, it wasn't needed, and decreased accuracy. The reason for this might be due to incredibly low resolution. Also a different layer is applied here. A dropout layer. This layer is used to prevent over-fitting the model. It randomly selects input cells which get erased to zero. Also it scales unaffected cells so the sum across the layer doesn't change. This increased accuracy quite well.

Convolutional neural networks configurations that well tried when prototyping were: Number of filter in convolutional layer, size of the filter kernel, dropout values in dropout layer.

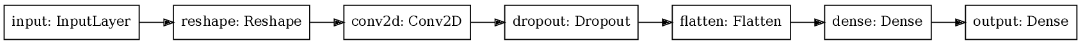


Figure 3.11: A general graph of Convolutional Neural Network used.

Recurrent Neural Network

For recurrent neural network, a LSTM (Long Short-Term Memory) layer is added.

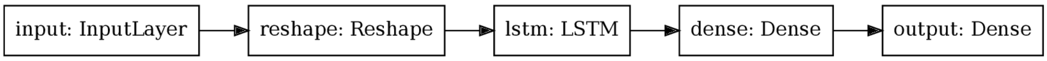


Figure 3.12: A general graph of Recurrent Neural Network used.

3.2.2 Shallow learning models

This subsection talks about prototyping shallow learning models. Shallow learning models are models that require a human to perform feature extraction on the dataset unlike deep learning models that accept mostly raw data.

Shallow learning models are implemented using a Scikit-learn (sklearn) library. Scikit-learn provides supervised, unsupervised machine learning algorithms and statistical models, as-well as some validation, dimensionality reduction, ensemble and feature-related models.

Training data

Shallow learning models require feature extraction steps prior to learning. This requires a feature selection part in process of developing a model. Random forests model is a little bit exception, it is capable of feature selection by itself. However there is one problem with Random Forests that makes it still require someone to select features. The problem is caused by feature correlation. Random Forests internal feature selection doesn't account for correlation in features being fed to it. Using correlated features makes Random Forest less accurate, most probably because it distorts feature importances, which is the foundation of Decision Trees. For this reason, Random Forests will be fed on selected features too.

Unlike deep learning models, shallow learning models will use unaugmented data. The reason for this is that their ideal amount of data is way lower than deep learning's, and it is more difficult to control over-fitting.

These features were explored for selection:

- Minimum value (min)
- Maximum value (max)
- Mean
- Variance (var)
- Sum
- Peak to peak value (ptp)
- Standard deviation (std)
- Sum across diagonals (trace)
- Marginal mean across all pixels on axis x (mmeanx)
- Marginal mean across all pixels on axis y (mmeanx)
- Marginal mean across high density cluster on axis x (mmeanxTF)
- Marginal mean across high density cluster on axis y (mmeanyTF)
- Marginal standard deviation across all pixels on axis x (msdx)
- Marginal standard deviation across all pixels on axis y (msdy)
- Marginal standard deviation on axis x using marginal mean across high density cluster (msdxTF)
- Marginal standard deviation on axis y using marginal mean across high density cluster (msdyTF)

This is similar to features explored for data elimination, but marginal means and standard deviations are added. These marginal functions show higher importance than all previous features as seen on figure

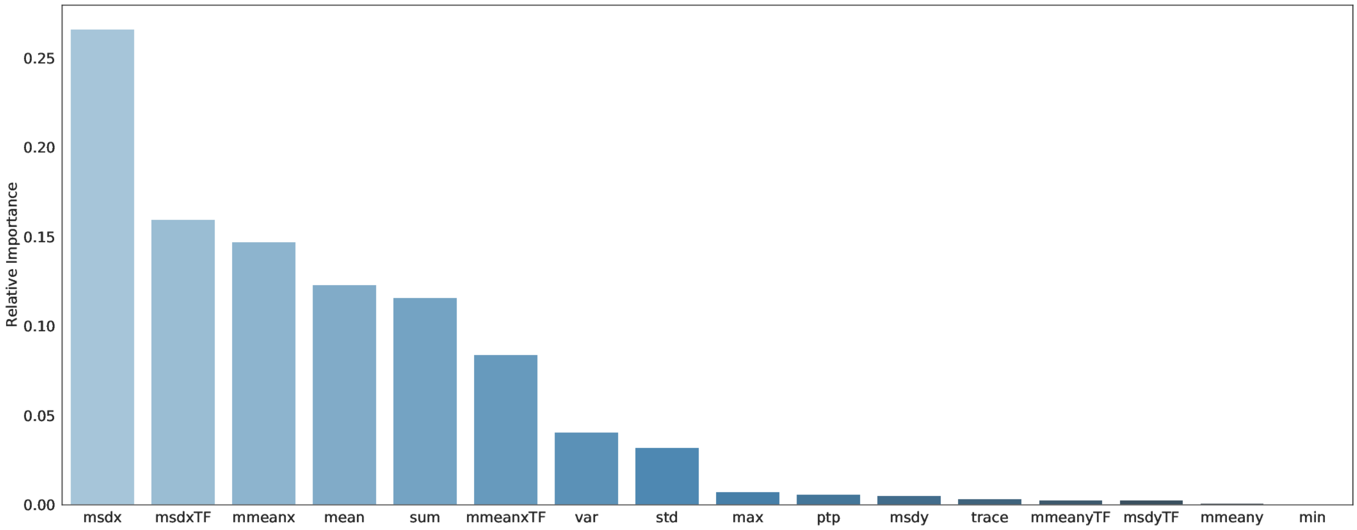


Figure 3.13: Random forest's feature importances.

Same as with data elimination. A Random Forest algorithm is used to determine feature importances, plotted graph is showed on figure 3.13. Where marginal standard deviation across axis x had stronger importance than all other features, which is a reason why I chose this feature as a first feature for learning.

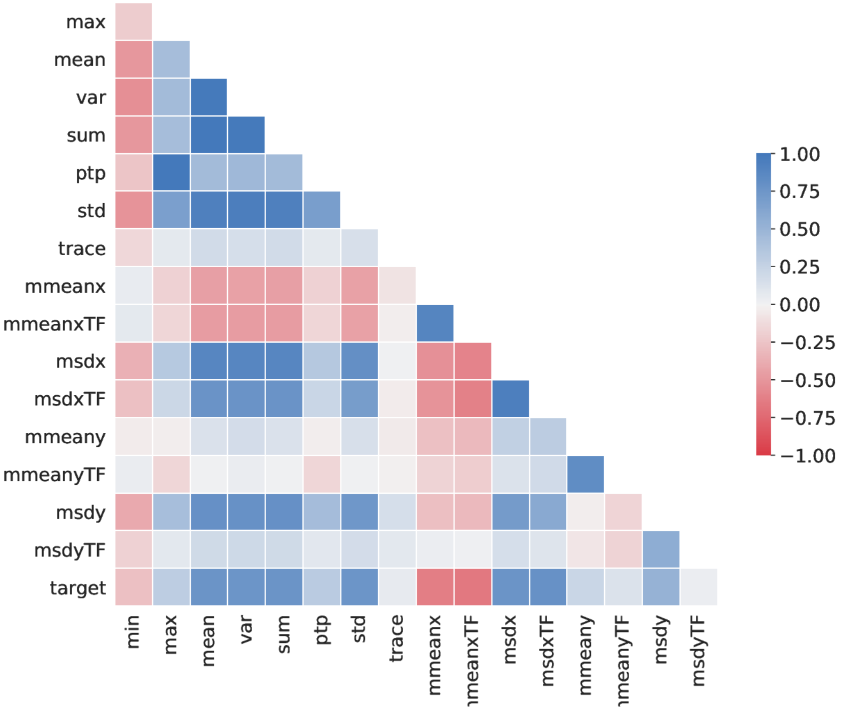


Figure 3.14: Correlation between features.

And similarly as in data elimination, a correlation plot, showed on figure 3.14. First feature that was chosen was „msdx“, and other features selected cannot have high correlation with this feature, again, above 0.9 or -0.9. For this reason, another features used are „mmeanxTF“ and „std“. Resulting features however, ended up being „msdxTF“, „mmeanx“, and „std“. This will be detailed in the next chapter.

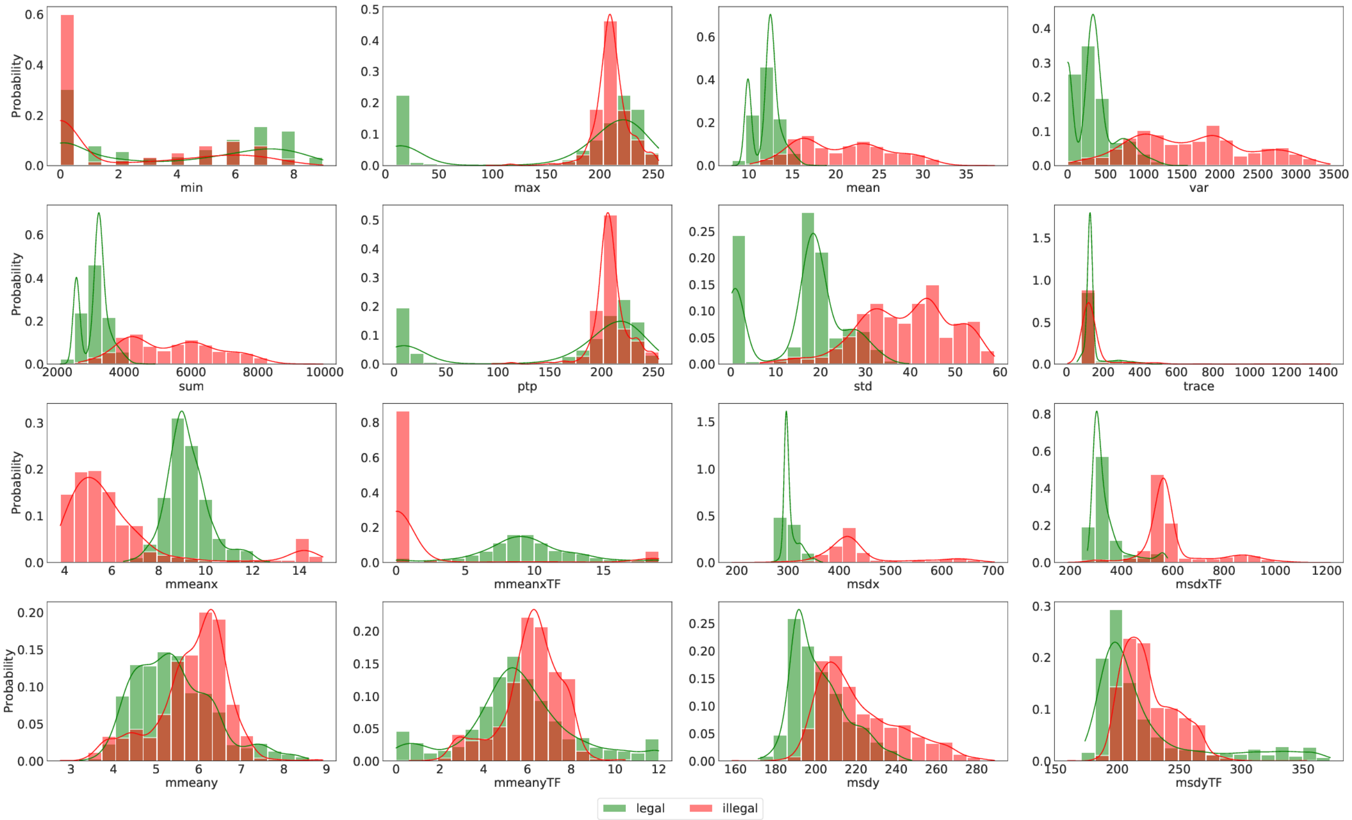


Figure 3.15: Probability distributions of each feature.

Three models are used for prototyping, all implemented using sklearn library:

- Logistic Regression
- Support Vector Machine
- Random Forests

3.3 Proof of concept

This section will talk about implementation of chosen model. To differentiate between prototypes and chosen model. This model will work real-time, and will show it’s input and output on graphical user interface.

Currently, the model uses Random Forest with three features. For user interface, library „Tkinter“ is used. To show the raw data of touchpad, figure from Seaborn library is fitted onto a canvas into the interface. To reduce loading times. Application uses serialized model from file, which is already trained.

All previous implementation used datasets gathered by a raw data collection program written in C. However, this demostration is written in Python, mainly due to machine learning libraries I was already familiar with. The original idea was using a Ctypes library for it to access the C code. However this proved to be too complex, mainly because of extensive structures that the C code uses. For this reason, a different approach was chosen: implementing V4L2 communication in python code itself. There is a v4l2 library available for Python, but last time it was updated was 2010, as the result, the code doesn't work on Python 3. For this reason, I had to search for a fork on GitHub.

Using a code from forked repository, I reimplemented V4L2 code in Python. Implementation was mostly similar to the one in C, however there was on problem when initiating and closing stream. Where the ioctl command requires an address for an integer value corresponding to buffer type. This cannot be done in Python well and was a large obstacles to making v4l2 code work. This was resolved by creating an array with this value, using `buf_type = array.array('I', [bufinfo.type])`. Other than that, v4l2 code is straightforward. This application still requires a recompiled and properly initiated driver.

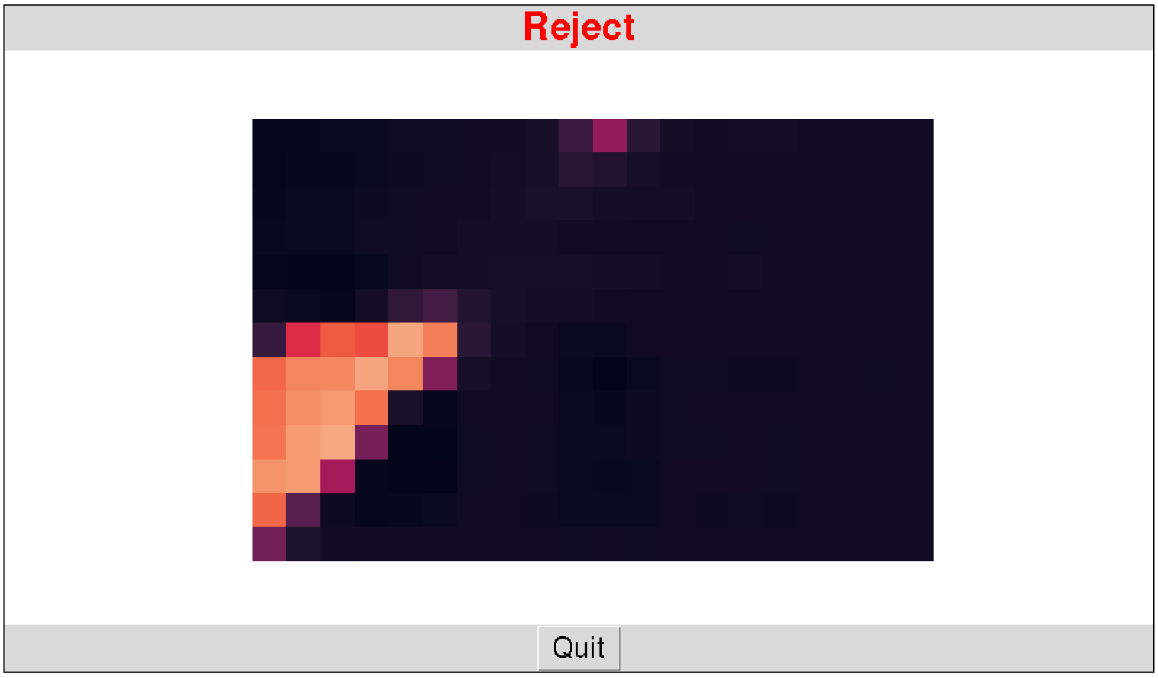


Figure 3.16: Screenshot of demostration UI, containing a visualisation of raw data and result of classification.

Chapter 4

Testing and summarization

The purpose of this chapter is to examine the behavior of models implemented in this thesis. Mainly their accuracy, memory, and time complexity. Accuracy is a quotient of correctly predicted labels to total amount of labels, as shown in equation 4.1. Memory complexity indicates how much computer memory algorithm utilizes. Time complexity suggests how much time is required to finish the calculation.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{4.1}$$

where:

- TP stands for true positive
- TN stands for true negative
- FP stands for false positive
- FN stands for false negative

4.1 Dataset

This section considers datasets used for training and testing and what possible effects they have on model behaviors. The structure of the training dataset and amount of images in them is as follows:

- Legal
  - Orig - 2204
  - Mirrored - 2155
  - Shifted - 180400
  - Shifted-mirrored - 180400
  - Rotated - 2155 each
- Illegal
  - Orig - 1602
  - Mirrored - 1565

  - Shifted - 5438
  - Shifted-mirrored - 5438
  - Rotated - 1565 each

The rotation angle can be almost any number, but only -5 degrees and +5 degrees are provided with the dataset. All the data augmentation tools are available with the thesis, especially another rotation degrees can be generated using program described in 3.1.3. Augmented data are used only for neural networks due to their requirements. However, they are tested with only `orig` data too. With more ideal conditions for collecting data, deep learning models could be experimented with using only original data; however, this is not the case for this thesis.

For reasons detailed in sub-section 3.1.2, data elimination is performed in the case of illegal data, which removes a portion of the dataset. About six percent of the illegal data has been removed due to this data elimination. The data elimination process removes chunks of data from the dataset, which may remove context from the data sequence required for recurrent neural networks. This raises questions about whether this dataset is ideal for RNNs, and whether it would perform better on datasets collected in a different way. The same concern is with part of data augmentation - shifting, which breaks the order of the images.

There are 378 images for the legal class and 236 images for the illegal class in the testing dataset. Those data are from new data collection scenarios. Images in this dataset are saved as .png files. The width of one image is 20 pixels, and its height is 13. Values range from value 0 to 255. Values from 0 to 9 correspond to negative values in the raw dataset, clipped down to a range of size 10. The reason for cutting down values like that is because, in some cases, negative values appear, but it seemed like there isn't a significant difference between a value -10 and -100. And because of the large size of the dataset, its 8-bit values should decrease dataset sizes.

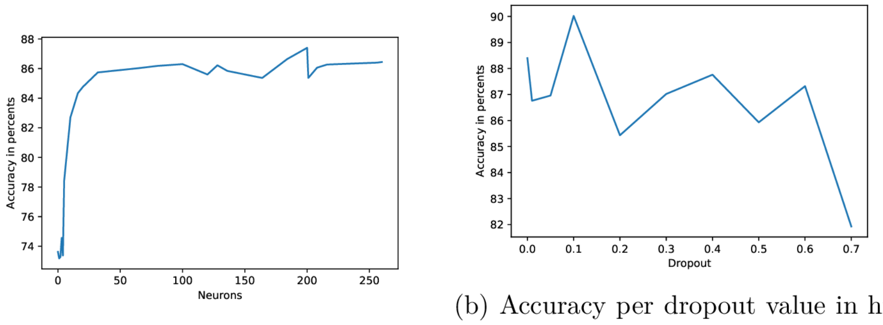
This dataset isn't representative of the problem since it is captured only by one person, me, due to the pandemic, which may introduce biases into the dataset and the testing dataset being too artificial and not having many extreme cases.

4.2 Neural Networks parameters

This section will look up how some configurations of neural networks affect their accuracy. In order to calculate accuracy, the model is trained and evaluated at least five times. Consequently, x-axis values are chosen randomly, without any planning, and this can cause graphs in this section to be biased. However, the purpose of this section is to explore networks behaviors, not analyze them.

4.2.1 Artificial neuron networks

For ANNs, different amounts of Relu neurons were experimented with. Surprisingly, even with one neuron, the model's accuracy was around 67%. Although this is low, it shows that neurons can work on their own.

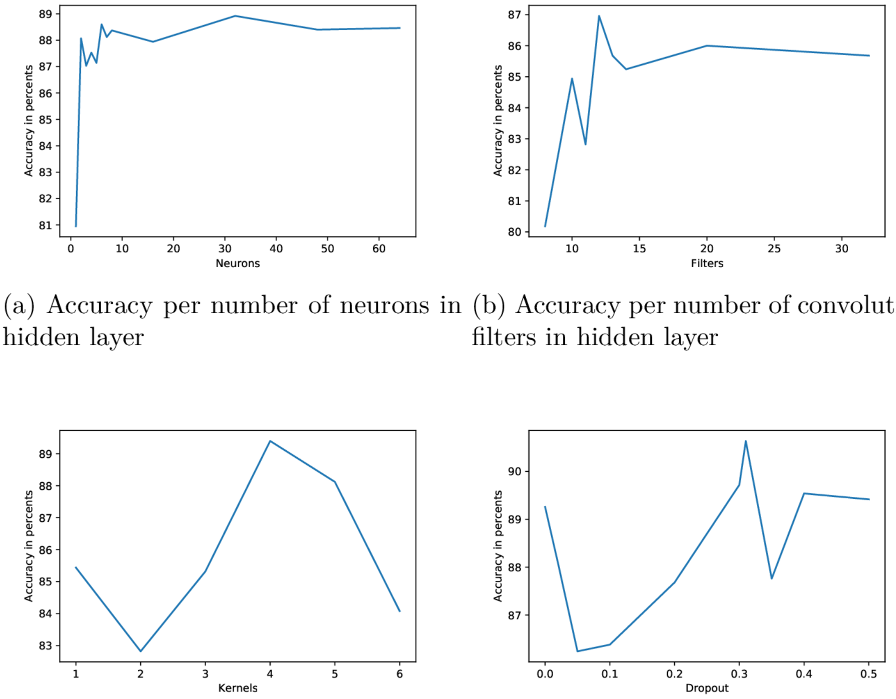


(a) Accuracy per number of Relu neurons in hidden layer.

(b) Accuracy per dropout value in hidden layer using 200 neurons in hidden layer.

4.2.2 Convolutional neuron networks

Amount of neurons were investigated using 12 filters and a 4x4 kernel. Dropout regularization was investigated using a 4x4 kernel, the layer of 6 neurons, and 14 filters. The number of convolutional filters was investigated using a 2x2 kernel. Kernels were investigated using 12 filters.



(a) Accuracy per number of neurons in hidden layer

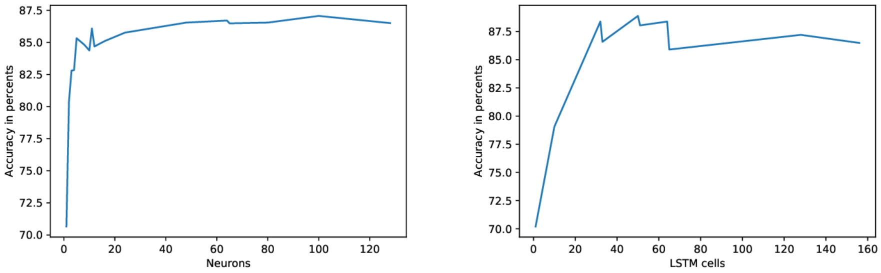
(b) Accuracy per number of convolution filters in hidden layer

(c) Accuracy per number size of the convolution kernel in hidden layer

(d) Accuracy per dropout value in hidden layer

4.2.3 Recurrent neural networks

In the hidden layer, 64 LSTM cells were used to investigate the accuracies of the numbers of neurons. 10 neurons were used in the following layer to explore the number of LSTM cells.



(a) Accuracy per number of neurons in hidden layer.

(b) Accuracy per number of LSTM cells in hidden layer.

4.3 Prototypes analysis

This section will talk about the behaviors of implemented models. Their accuracies, time complexity, and for few, behaviors when changing their parameters (such as for deep learning the neural network's shape). However, this is not a proper analysis and should be used for comparing models and possibly for trying to understand how they work.

To get close to shallow models' accuracy, neuron networks were fed with a bigger dataset to reach accuracy above 90%. Deep learning models were fed with a training dataset containg 19,371 images, containing datasets `orig`, `mirrored`, `rotated5.0` and `rotated-5.0` as opposed to 3,806 images used for shallow models, made up by only an `orig` dataset.

4.3.1 Accuracy

Logit is using features: `mmeanxTF`, `mean`, `msdxTF`. Random forests features `mmeanx`, `std`, mstdxTF. SVM is using all features shown in subsection 3.2.2. For deep learning models, accuracy is again calculated from 5 measurements. Accuracies are calculated from predictions of a new dataset of 614 images.

Deep learning model following accuracies were measured: 94.59% for ANN, 95.86% for CNN, 92.93% for RNN. For logistic regression, resulting accuracy was 98.04%, for SVM classifier 99.35% and for random forests 99.51%.

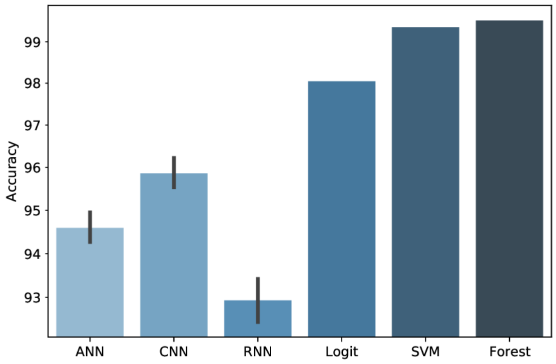
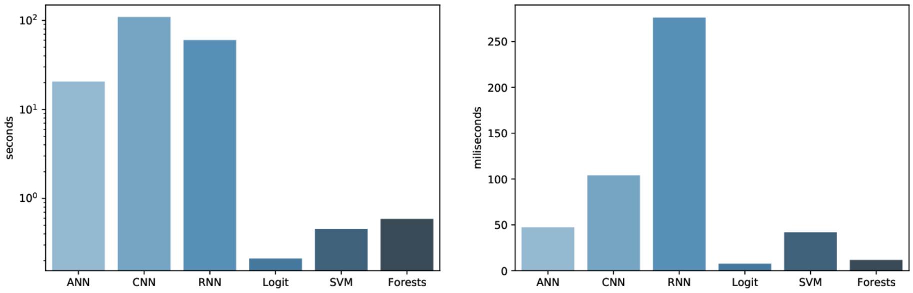


Figure 4.4: Prototype accuracies

4.3.2 Time complexity

Time duration of algorithms are measured by a `%time` profiler provided with IPython. This measures CPU user time, kernel time, and real time, from which CPU user times are measured.

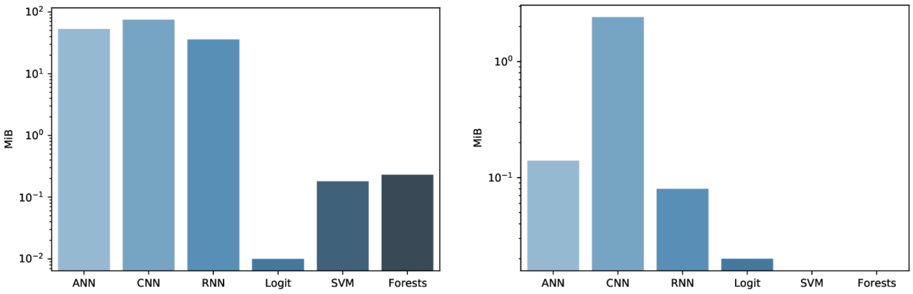


(a) Time duration when training.

(b) Time duration when predicting across a dataset.

4.3.3 Memory complexity

Memory usage is measured by a `%memit` profiler provided with IPython. This provides total memory used by the kernel, and the difference in memory budget before and after running the command, which were used to calculate memory usages.



(a) Memory usage when training.

(b) Memory usage when predicting across a dataset.

4.4 Summarization

From all prototypes, the best accuracy was observed with the Random Forests algorithm. Which also has an exquisite time complexity and memory complexity when predicting labels. For this reason, it was chosen as the best model for demonstration application. Most models were run with standard parameters, and there might be better combinations of them. However, to show comparisons between each algorithm, this should be accurate enough. Also, there might be a lot more promises for neural networks in terms of accuracy. However, when run on a CPU, their time and memory complexity when predicting labels is considerably higher. This leaves the possibility that if neural networks were modeled and trained better, their accuracy might be higher in more extreme cases. For neural calculations, however, a special instrument would be required due to time and memory constraints.

Models expectations

Before starting the thesis, it was thought that data that can be extracted from touchpad are not good enough to have any successful accuracy. Which in the end was not true, since there were many models explored with accuracy above 90%. My expectation was that shallow learning methods will have accuracy lower than 90%. And that I will reach accuracy above 90% only if I will be able to train a neural network with large enough dataset. This again, didn't happen, since shallow learning methods were reaching very good accuracies, over 99%.

Comparison to existing solutions

Existing solutions were explored in subsection 2.2.2. Where Annet et. al. [2] showed several rejection methods, from which the most accurate one was a hover detection with accuracy below 90%. And [18] reached 97.9%. Those methods, however are for touchscreens, which is slightly different. Regardless results show that result accuracies are very high compared to other publicly existing solutions.

Chapter 5

Conclusion

In this thesis, three criteria were used to compare various AI/ML models for palm detection: accuracy, speed and resource insensitivity. The chosen model is Random Forests. Results showed higher accuracy than models it was compared to. But time and memory complexity of those methods wasn't known, and thus it couldn't be compared with the proposed model. Due to low time and memory complexity, the chosen model could be implemented in the firmware of the touchpad device and provide palm detection without adding significant latency. Beyond those three criteria, a large limitation was found, mainly caused by how laptops are built. Where touchpads are not designed to send raw data through the bus, it is connected through. Because of this, real-time demonstration of the selected model makes cursor movements stutter; the touch-pads bus cannot handle raw data transmission and makes the bus overloaded. This limitation, however, doesn't apply to implementing such a model in the firmware by laptop manufacturers.

This thesis can be continued upon by better adjustment of models parameters. Exploring different models, for example, Hopfield Networks (HNNs). Creating a bigger, unaugmented, and better-representing dataset, presumably by a survey from multiple people, by representative study, without the need to use data augmentation. Designing a better neural network structure. Introduce new data than just raw data.

This thesis will hopefully motivate companies to explore AI/ML for touch rejection, publicly and open-source. Or give developers access to touchpad data to implement palm rejection in kernel space because the current state of touchpads is a major limitation for subsequent research.

Bibliography

[1] Abdiansah, A. and Wardoyo, R. Time complexity analysis of support vector machines (SVM) in LibSVM. *International journal computer and application*. Taylor & Francis. 2015, vol. 128, no. 3, p. 28–34.

[2] Annett, M., Gupta, A. and Bischof, W. F. Exploring and understanding unintended touch during direct pen interaction. *ACM Transactions on Computer-Human Interaction (TOCHI)*. ACM New York, NY, USA. 2014, vol. 21, no. 5, p. 1–39.

[3] Anonymous. *What is a Random Forest?* Tibco. https://www.tibco.com/reference-center/what-is-a-random-forest.

[4] Anthony, M. and Bartlett, P. L. *Neural network learning: Theoretical foundations*. Cambridge university press, 2009.

[5] Biau, G. Analysis of a random forests model. *The Journal of Machine Learning Research*. JMLR. org. 2012, vol. 13, p. 1063–1095.

[6] Breiman, L. Random forests. *Machine learning*. Springer. 2001, vol. 45, no. 1, p. 5–32.

[7] Camilleri, M. J., Malige, A., Fujimoto, J. and Rempel, D. M. Touch displays: the effects of palm rejection technology on productivity, comfort, biomechanics and positioning. *Ergonomics*. Taylor & Francis. 2013, vol. 56, no. 12, p. 1850–1862.

[8] DeMaris, A. A tutorial in logistic regression. *Journal of Marriage and the Family*. JSTOR. 1995, p. 956–968.

[9] Donges, N. *A complete guide to the random forest algorithm*. Built In, June 2019. https://builtin.com/data-science/random-forest-algorithm.

[10] Erdelyil, A., Sihvonen, T., Helin, P. and Hänninen, O. Shoulder strain in keyboard workers and its alleviation by arm supports. *International archives of occupational and environmental health*. Springer. 1988, vol. 60, no. 2, p. 119–124.

[11] Hilbe, J. M. Logistic Regression. *International encyclopedia of statistical science*. 2011, vol. 1, p. 15–32.

[12] Hornik, K., Stinchcombe, M. and White, H. Multilayer feedforward networks are universal approximators. *Neural networks*. Elsevier. 1989, vol. 2, no. 5, p. 359–366.

[13] Kratsios, A. and Bilokopytov, E. Non-euclidean universal approximation. *ArXiv preprint arXiv:2006.02341*. 2020.

[14] Mohri, M., Rostamizadeh, A. and Talwalkar, A. *Foundations of machine learning*. MIT press, 2018.

[15] Raschka, S. *Kernel tricks and nonlinear dimensionality reduction via RBF kernel PCA*. September 2016.
https://sebastianraschka.com/Articles/2014_kernel_pca.html.

[16] Reynolds, D. A. Gaussian mixture models. *Encyclopedia of biometrics*. Springer City. 2009, vol. 741, p. 659–663.

[17] Schnyer, D. M., Clasen, P. C., Gonzalez, C. and Beevers, C. G. Evaluating the diagnostic utility of applying a machine learning algorithm to diffusion tensor MRI measures in individuals with major depressive disorder. *Psychiatry Research: Neuroimaging*. Elsevier. 2017, vol. 264, p. 1–9.

[18] Schwarz, J., Xiao, R., Mankoff, J., Hudson, S. E. and Harrison, C. Probabilistic palm rejection using spatiotemporal touch features and iterative classification. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2014, p. 2009–2012.

[19] Shanis, J. M. and Hedge, A. Comparison of mouse, touchpad and multitouch input technologies. In: SAGE Publications Sage CA: Los Angeles, CA. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. 2003, vol. 47, no. 4, p. 746–750.

[20] Shneiderman, B. Touch screens now offer compelling uses. *IEEE software*. IEEE. 1991, vol. 8, no. 2, p. 93–94.

[21] Tripathi, M. *Basic Overview of SVM Algorithm*. DataScience foundation, December 2020. https://datascience.foundation/datatalk/basic-overview-of-svm-algorithm.

[22] Vogel, D., Cudmore, M., Casiez, G., Balakrishnan, R. and Keliher, L. Hand occlusion with tablet-sized direct pen input. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2009, p. 557–566.

[23] Walker, G. Palm rejection on resistive touchscreens. *Veritas et Visus*. 2005, p. 31–33.