ABSTRACT

The feature extraction phase of an offline signature verification system is regarded as crucial and has a significant impact on the performance of these systems, as the quantity and calibration of the extracted features determine how well these systems can differentiate between authentic and fake signatures. It introduces a hybrid method for feature extraction from signature images in offline signature verification systems. The method uses a combination of Convolutional Neural Network (CNN) and Histogram of Oriented Gradients (HOG) techniques, followed by a feature selection algorithm (Decision tree) to identify key features. The hybrid method was evaluated using classifiers (Like support vector machine, and K-nearest Neighbor) and tested on two datasets (UT Sig and CEDAR). The experimental results showed high accuracy in distinguishing between authentic and forged signatures, even for skilled forgeries.

INDEX

LIST OF FIGURES

LIST OF SCREENS

1. INTRODUCTION

Biometrics represents the most important technological method used to identify people and determine their power through the behavioral and physiological characteristics of individuals. The handwritten signature is one of the most accepted methods of biometric verification in the world. Banks, credit cards, passports, check processing, and financial documents use handwritten signatures as unique behavioral biometrics. It is difficult to verify these signatures, particularly when they are unclear. Therefore, a system that can distinguish between a genuine signature and a fake signature is required to lower the chance of theft or fraud. In the past thirty years, several studies have been conducted in this field, from traditional verification based on expert opinions to machine learning algorithms, then deep learning algorithms today, despite all these studies, offline signature verification systems still need a lot of development and improvement. Based on the discussion, the current study aims to develop a hybrid method for extracting features from signature images and classifying them for authentic and forged signatures using deep learning and machine learning classifiers to ensure that the hybrid method can enhance the performance of various classifiers. This method will be used in offline signature verification.

Signature verification is a critical task in various fields, including banking, legal, and security sectors, where the authenticity of handwritten signatures plays a pivotal role in ensuring document integrity and preventing fraud. Traditional methods for signature verification often rely on handcrafted features and heuristic algorithms, which may struggle to capture the intricate details and variations present in handwritten signatures. In recent years, the emergence of deep learning techniques, particularly Convolutional Neural Networks (CNN), has revolutionized the field of pattern recognition, offering unparalleled capabilities in learning complex representations directly from raw data. However, CNNs may exhibit limitations when applied to tasks involving fine-grained texture and shape analysis, such as signature verification. On the other hand, Histogram of Oriented Gradients (HOG) is a classical method that excels in capturing local texture and shape information, making it well-suited for signature analysis. In this context, the hybrid method for feature extraction using CNN and HOG presents a compelling approach to overcome the limitations of individual methods and enhance the accuracy and robustness of signature verification systems. By combining the high-level abstract features learned by CNNs with the detailed texture and shape descriptors computed by HOG, the hybrid method leverages the complementary strengths of both techniques to create a comprehensive representation of handwritten signatures. This introduction sets the stage for exploring the methodology, applications, and potential benefits of the hybrid CNN-HOG approach for signature verification, highlighting its relevance in addressing real-world challenges and advancing the state-of-the-art in document authentication and fraud detection.

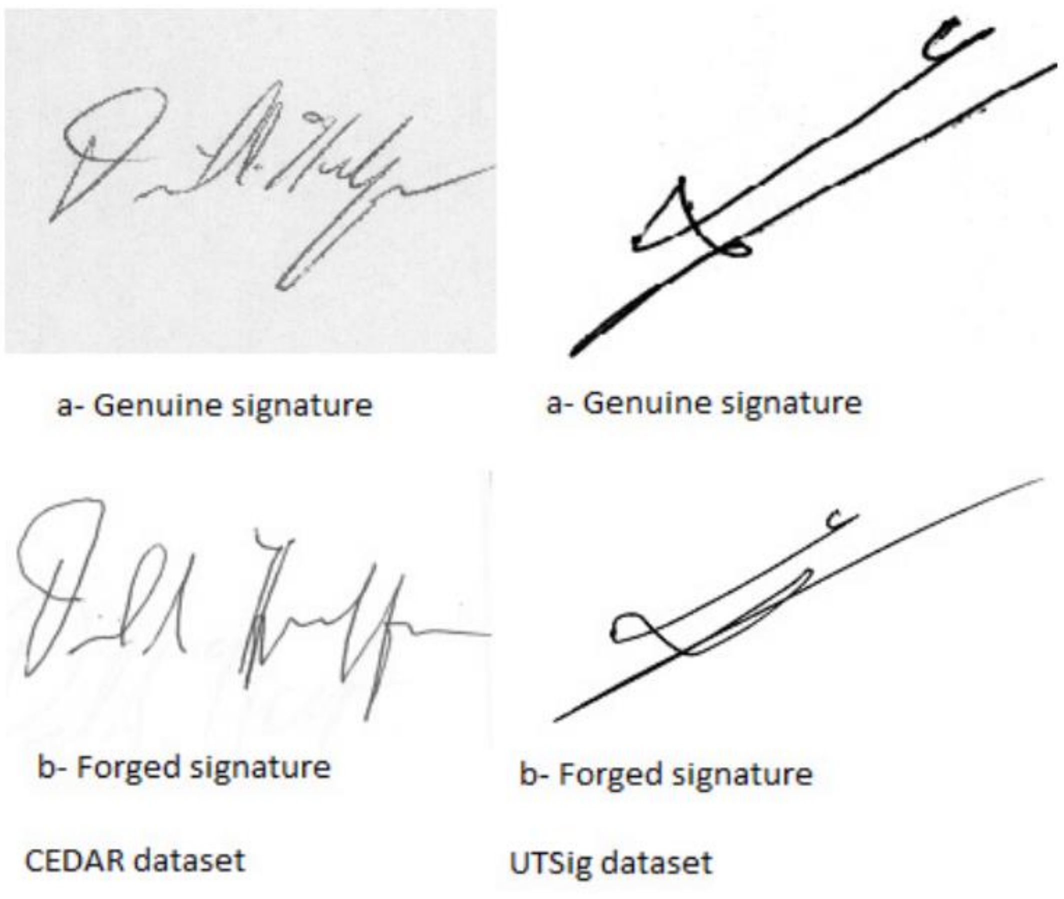


Figure 1: Signature Dataset

1.1 OBJECTIVE

The objective of this research paper is to propose and evaluate a hybrid feature extraction method for offline signature verification systems, combining Convolutional Neural Network (CNN) and Histogram of Oriented Gradients (HOG) techniques, followed by feature selection using a Decision Tree algorithm. The goal is to enhance the accuracy of distinguishing between genuine and forged signatures, particularly skilled forgeries, using a variety of classifiers on two distinct datasets (UTSig and CEDAR).

By merging the capabilities of both methods, this approach achieves a more comprehensive representation of visual data. Initially, the input image undergoes preprocessing steps to enhance its quality and suitability for feature extraction. Subsequently, a pre-trained CNN model is employed to extract high-level features, capturing intricate patterns, textures, and shapes. Concurrently, HOG computes gradient orientation histograms within localized image regions, capturing fine-grained details and local texture information. Through feature fusion, the high-level abstractions from the CNN and the detailed descriptors from HOG are combined, forming a hybrid feature representation that encapsulates both semantic richness and local intricacies. This fusion results in enhanced robustness to variations in illumination, scale, and viewpoint, ultimately improving the accuracy and generalization capability of computer vision tasks such as object detection, recognition, and classification. The approach leverages the complementary strengths of CNNs and HOG, providing a balanced and effective solution for image analysis in various domains.

1.2 EXISTING SYSTEM:

In existing works, offline verification of signatures is carried out using a set of simple-shaped geometric features. These characteristics include the Baseline Slant Angle (BSA), the Aspect Ratio (AR), and the Normalized Area (NA), as well as the Center of Gravity and the Slope of the line linking the two Centers of Gravity of the signature's image segments. Initially, System training is carried out through a record of signatures obtained from individuals whose signatures had to be validated by the system. Therefore, a signature functions as the prototype for authentication against a requested test signature. The similarity measure within the feature space between the two signatures is determined by Euclidian distance. If the Euclidian distance is lower than a set threshold (i.e. analogous to the minimum acceptable degree of similarity), the test signature is certified as that of the claiming subject otherwise detected as a forgery. Details on the stated features, pre-processing, implementation, and the results are presented in this work

HOG feature extraction technique focuses on capturing the distribution of gradient orientations within local image regions. By dividing the signature image into small cells and computing histograms of gradient orientations within each cell, HOG generates feature descriptors that encode shape and edge information. While HOG may lack the ability to capture high-level semantic features like CNNs, it excels at representing structural characteristics that are often informative for signature analysis.

In the hybrid method, the CNN and HOG features are extracted independently from the input signature images. The CNN extracts deep features directly from the pixel values, while the HOG algorithm computes feature descriptors based on gradient orientation histograms. These features are then concatenated or fused together to form a combined feature vector that preserves both high-level semantic information from the CNN and structural details from HOG.

After feature extraction, the hybrid system employs a classification model to differentiate between genuine and forged signatures. This model can be a traditional machine learning algorithm such as Support Vector Machines (SVM) or a more advanced deep learning architecture like a fully connected network or a recurrent neural network. The classification model is trained on a labeled dataset containing examples of genuine and forged signatures, using the combined feature vectors as input.

During the training phase, the parameters of both the CNN and the classification model are optimized jointly to minimize a suitable loss function, such as cross-entropy loss. This joint optimization allows the system to learn discriminative representations that effectively capture the nuances of genuine signatures while also being robust to various types of forgeries.

Once trained, the hybrid system can be deployed for real-world signature verification tasks. Given a new signature image, the system first extracts CNN and HOG features, combines them into a unified feature representation, and then feeds this representation into the trained classification model. The model outputs a probability score indicating the likelihood that the input signature is genuine. Based on this score, a decision threshold can be applied to classify the signature as either genuine or forged.

In summary, the hybrid method for signature verification using CNN and HOG combines the complementary strengths of deep learning and traditional image processing techniques. By integrating CNN's ability to capture semantic features with HOG's proficiency in encoding structural information, the hybrid system achieves enhanced performance in distinguishing between genuine and forged signatures, making it a valuable tool for various applications requiring reliable authentication and document security.

1.2.1 DISADVANTAGES OF EXISTING SYSTEM:

- The approach described in the existing works relies on a limited set of geometric features. While these features may provide some discriminatory power, they may not capture all the relevant information necessary for accurate and robust signature verification. Additional complex or texture-based features could potentially improve the system's performance.
- The effectiveness of the signature verification system heavily depends on the quality of the signature preprocessing step, which segregates parts and eliminates noise. If the preprocessing stage is not robust enough or fails to address all potential variations or noise types, it may adversely affect the accuracy and reliability of the feature extraction process.
- The prototype-based authentication method may have limited generalization ability when faced with new signatures or signatures from different individuals.

1.3 Proposed System:

In this study, we introduced a hybrid method for feature extraction from signature images using a combination of Convolutional Neural Network (CNN) and Histogram of Oriented Gradients (HOG) techniques, followed by feature selection algorithm using decision trees to identify the key features. The aim is to improve the accuracy and efficiency of offline signature verification systems by identifying key features that can distinguish between authentic and fake signatures. The developed method was evaluated using three classifiers (long short-term memory, support vector machine, and K-nearest Neighbor) on two datasets (UTSig and CEDAR).

The proposed system begins with the preprocessing stage, where signature images are preprocessed to enhance their quality and facilitate feature extraction. This preprocessing may include operations such as noise removal, normalization, and resizing to ensure consistency across different samples.

Next, the system extracts features from the preprocessed signature images using both CNN and HOG techniques. The CNN component is responsible for learning hierarchical representations of signatures directly from the raw pixel data. Through multiple layers of convolution, activation, and pooling, the CNN captures intricate patterns and features within the signatures, including stroke width, texture variations, and spatial relationships.

Simultaneously, the HOG algorithm computes feature descriptors that encode the distribution of gradient orientations within local image regions. By dividing the signature image into small cells and computing histograms of gradient orientations, HOG generates feature representations that capture shape and edge information, which are crucial for signature analysis.

Once the CNN and HOG features are extracted independently, they are fused together to create a unified feature representation that preserves both high-level semantic information from the CNN and structural details from HOG. This fusion process may involve concatenating the feature vectors or employing more sophisticated techniques such as feature-level or decision-level fusion.

With the combined feature representation in hand, the system utilizes a classification model to differentiate between genuine and forged signatures. This model can be a traditional machine learning algorithm like Support Vector Machines (SVM) or a deep learning architecture such as a fully connected network or a convolutional neural network.

During the training phase, the parameters of both the CNN and the classification model are optimized jointly to minimize a suitable loss function, such as cross-entropy loss or hinge loss. This joint optimization enables the system to learn discriminative representations that effectively capture the nuances of genuine signatures while remaining robust to various types of forgeries.

Once trained, the hybrid system is ready for deployment in real-world signature verification tasks. Given a new signature image, the system first preprocesses the image, extracts CNN and HOG features, and then fuses them into a unified representation. This representation is then fed into the trained classification model, which outputs a probability score indicating the likelihood that the input signature is genuine.

Based on this score, a decision threshold can be applied to classify the signature as either genuine or forged, enabling reliable authentication and document security in various applications.

In summary, the proposed hybrid method for signature verification using CNN and HOG offers a powerful solution for enhancing the accuracy and robustness of signature verification systems. By combining the strengths of deep learning and traditional image processing techniques, the proposed system achieves superior performance in distinguishing between genuine and forged signatures, making it a valuable tool for applications requiring secure and reliable authentication.

1.3.1 Advantages of proposed system:

1. The hybrid model has a robust feature set and may work in conjunction with a low complexity classifier to improve performance.
2. The use of three classifiers from machine learning and deep learning will contribute to confirming the importance of the hybrid method adopted in extracting features.
3. The multi-classifier, multi-dataset evaluation enhances the robustness and generalizability of the proposed method.
4. Efficient Feature Extraction: CNNs are known for their ability to automatically learn hierarchical features from raw image data, while HOG features capture local gradient information. This combination can capture both global and local characteristics of signatures, resulting in more comprehensive feature representations.
5. By employing feature selection algorithms using decision trees, the study focuses on selecting the most discriminative features for classification. This approach can potentially lead to enhanced accuracy.

2.LITERATURE SURVEY

2.1 Signature Forgery Detection Using Machine Learning:

In today's society, signature is used many important documents such bank cheque, passport, driving license, etc. and can be faked in multiple ways. This creating many problems such as fake identifications, identify theft, hacking etc. To reduce this issue, our project is focused on developing a system for detecting whether a signature is real or fake from dataset of signatures using CNN and Deep learning. The reason we are using CNN and deep learning is because signature change over a period of time based on multiple behavioral changes such age, state of mind, physical health etc. We require a system that can learn from multiple training datasets and increase its accuracy of detection. There are two types of signatures authentication methods, which are online signature and offline signature verification methods. Our project is based on offline signatures forgery detection method. This type of signatures is handwritten on the documents and require an image of the signature. This is why we also should consider image processing for this project. We are referencing a few papers which implement the project using a few methods for both online and offline signature forgery detection methods based on deep learning models, we plan on implementing the offline methods and try to achieve a better accuracy.

2.2 Handwritten Signatures Forgery Detection Using Pre-Trained Deep Learning Methods:

Handwritten signature recognition (HSR) is crucial in various applications, such as document verification, authentication, financial transactions, banking transactions, and legal agreements. However, the prevalence of signature forgery poses a significant challenge to the integrity and security of these authentication systems. The purpose of signature forgery detection (SFD) systems is to discriminate between genuine signatures (by the purported person) and forged ones (by an impostor), which is a challenging task, especially in offline scenarios that use scanned signature images for signature recognition, where dynamic information about the signing process is not available. In recent years, pre-trained deep learning (DL) models have been widely used in image processing tasks due to their ability to achieve high accuracy with minimal training time and computational resources. By leveraging pre-trained models, developers can avoid starting from scratch when training a model, which can save time. Therefore, some pre-trained DL models are used for SFD in this paper and compared with each other. The result of implementing these methods shows that these methods have good accuracy for SFD. The MobileNet model, in particular, shows remarkable accuracy, reaching approximately 98.44%. In addition, it offers the advantages of relatively short training time and compact model size. These valuable features make MobileNet suitable for deploying mobile devices and embedded systems.

2.3 An integrated approach on verification of signatures using multiple classifiers (SVM and Decision Tree): A multi-classification approach:

A signature is a handwritten representation that is commonly used to validate and recognize the writer individually. An automated verification system is mandatory to verify the identity. The signature essentially displays a variety of dynamics and the static characteristics differ with time and place. Many scientists have already found different algorithms to boost the signature verification system function extraction point. The paper is aimed at multiplying two different ways to solve the problem in digital, manual, or some other means of verifying signatures. The various characteristics of the signature were found through the most adequately implemented methods of machine learning (support vector and decision tree). In addition, the characteristics were listed after measuring the effects. An experiment was performed in various language databases. More precision was obtained from the feature.

2.4 Recent developments in pretreatment technologies on lignocellulosic biomass: Effect of key parameters, technological improvements, and challenges:

Lignocellulosic biomass is an inexpensive renewable source that can be used to produce biofuels and bioproducts. The recalcitrance nature of biomass hampers polysaccharide accessibility for enzymes and microbes. Several pretreatment methods have been developed for the conversion of lignocellulosic biomass into value-added products. However, these pretreatment methods also produce a wide range of secondary compounds, which are inhibitory to enzymes and microorganisms. The selection of an effective and efficient pretreatment method discussed in the review and its process optimization can significantly reduce the production of inhibitory compounds and may lead to enhanced production of fermentable sugars and biochemicals. Moreover, evolutionary and genetic engineering approaches are being used for the improvement of microbial tolerance towards inhibitors. Advancements in pretreatment and detoxification technologies may help to increase the productivity of lignocellulose-based biorefinery. In this review, we discuss the recent advancements in lignocellulosic biomass pretreatment technologies and strategies for the removal of inhibitors.

2.5 Offline Handwritten Signature Verification Using Deep Neural Networks:

Prior to the implementation of digitisation processes, the handwritten signature in an attendance sheet was the preferred way to prove the presence of each student in a classroom. The method is still preferred, for example, for short courses or places where other methods are not implemented. However, human verification of handwritten signatures is a tedious process. The present work describes two methods for classifying signatures in an attendance sheet as valid or not. One method based on Optical Mark

Recognition is general but determines only the presence or absence of a signature. The other method uses a multiclass convolutional neural network inspired by the AlexNet architecture and, after training with a few pieces of genuine training data, shows over 85% of precision and recall recognizing the author of the signatures. The use of data augmentation and a larger number of genuine signatures ensures higher accuracy in validating the signatures.

3.SYSTEM REQUIREMENTS

3.1 SOFTWARE REQUIREMENTS

Software: Anaconda

Primary Language: Python

Frontend Framework: Flask

Back-end Framework: Jupyter Notebook

Database: Sqlite3

Front-End Technologies: HTML, CSS, JavaScript and Bootstrap4

Operating System: Windows 10

3.2 HARDWARE REQUIREMENTS

Processor: I5 and above

RAM: 8GB and above

Hard Disk: 25 GB

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE:

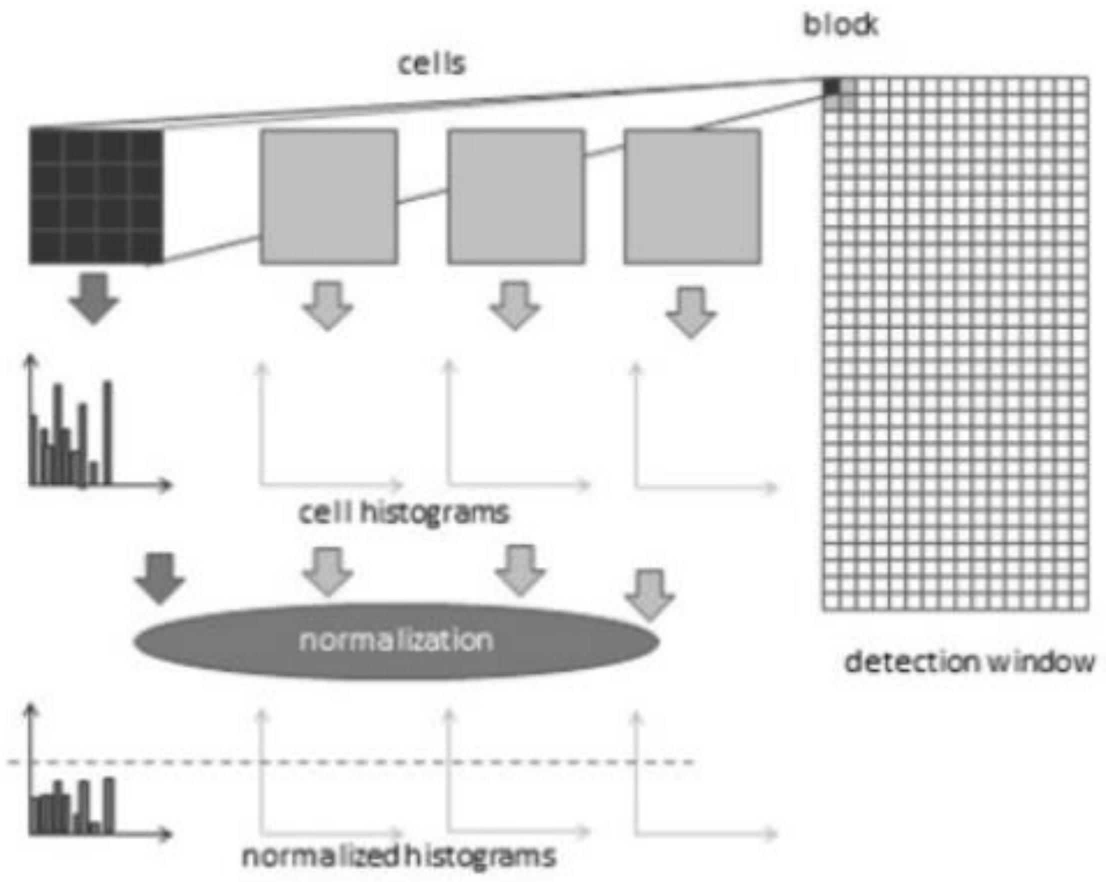


Figure 2: System architecture

DATA FLOW DIAGRAM:

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

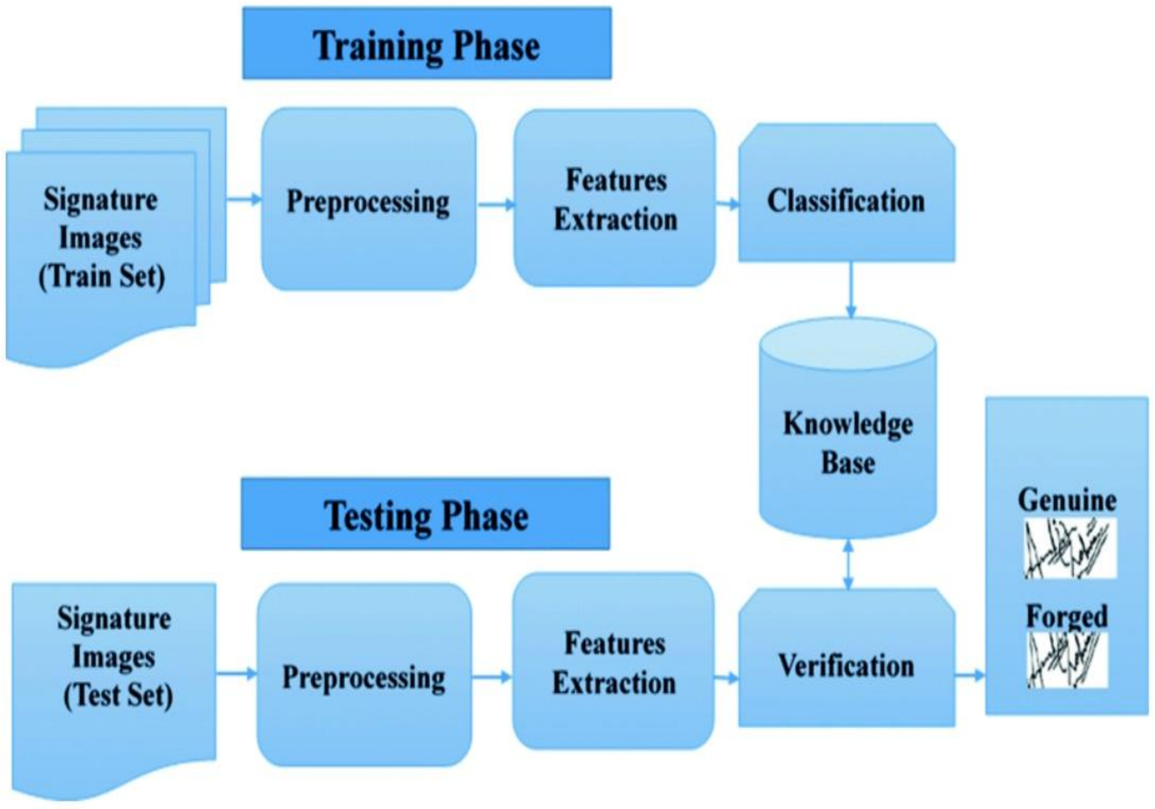


Figure 3: Flow diagram

4.2 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

4.2.1 Use case diagram:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

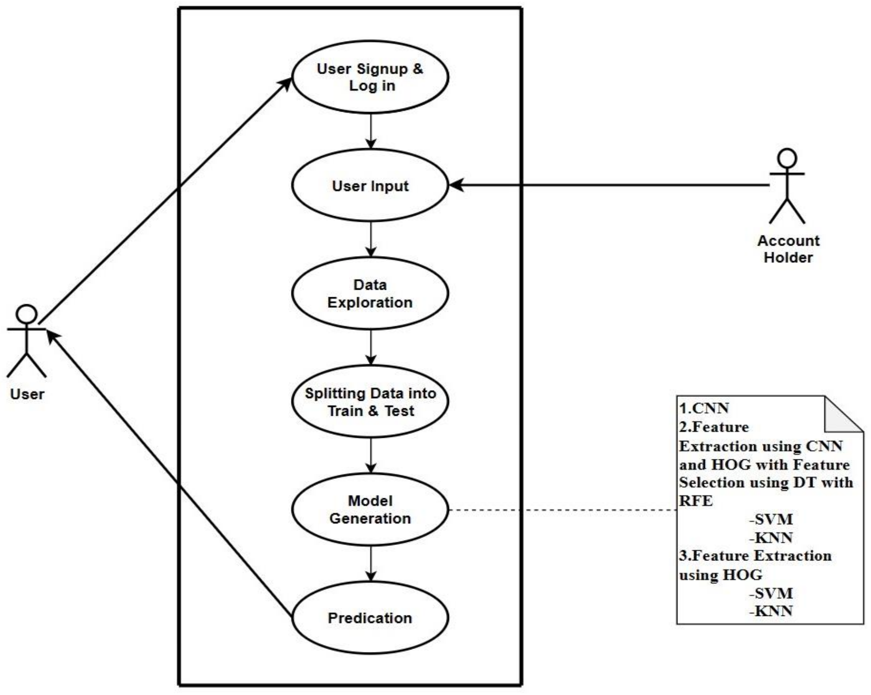

User Signup & Log in
User Input
Data Exploration
Splitting Data into Train & Test
Model Generation
Predication
User
Account Holder
1.CNN
2.Feature Extraction using CNN and HOG with Feature Selection using DT with RFE
-SVM
-KNN
3.Feature Extraction using HOG
-SVM
-KNN

Figure 4: Use case Diagram

4.2.2 Class diagram:

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

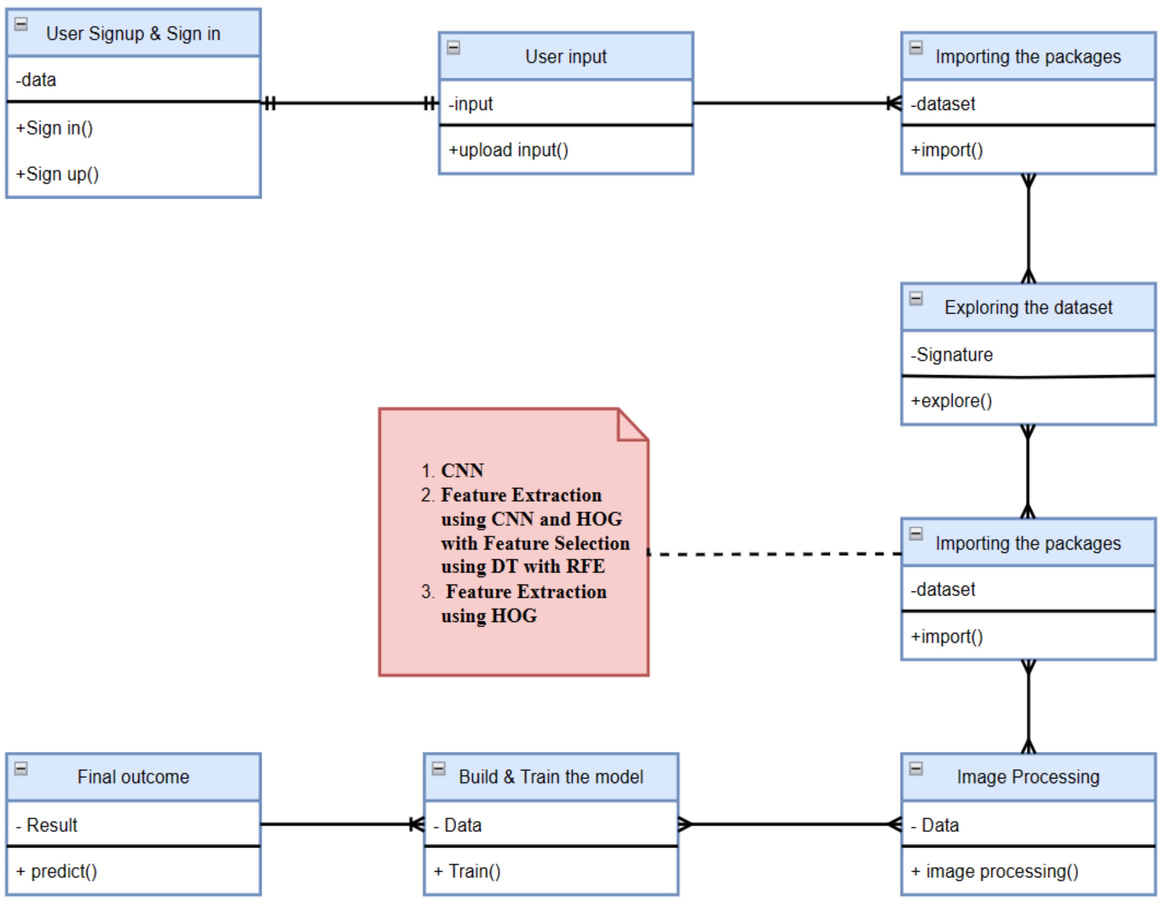


Figure 5: Class Diagram

Signature verification systems, the class diagram serves as a fundamental tool for refining the system's design, outlining its structure, and defining the interactions among its various components. At the core of this diagram lie the classes, each representing a distinct entity within the system, equipped with attributes that uniquely identify them and methods that enable them to perform specific functionalities. In the context of a hybrid signature verification system employing Convolutional Neural

Networks (CNN) and Histogram of Oriented Gradients (HOG), the class diagram plays a pivotal role in delineating the relationships between these components and orchestrating their collaboration towards the overarching goal of accurately verifying signatures.

4.2.3 Activity diagram:

The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions.

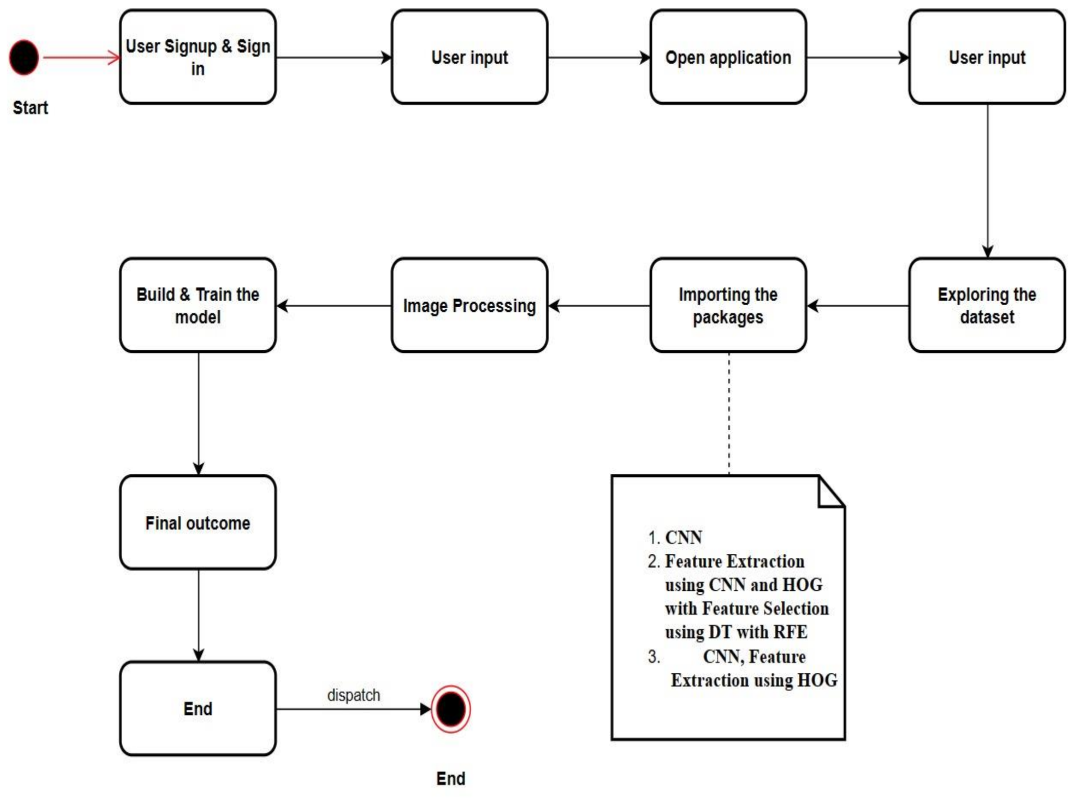


Figure 6: Activity Diagram

Activity diagrams serve as powerful tools for visualizing and understanding the flow of activities within a system, providing a detailed representation of how processes unfold and progress from one state to another. Much like state diagrams, activity diagrams comprise activities, actions, transitions, initial and final states, and guard conditions, each playing a crucial role in depicting the sequential and parallel execution of tasks within the system.

At the core of an activity diagram are activities, representing the high-level tasks or processes that occur within the system. These activities encapsulate units of work that contribute to achieving specific objectives or fulfilling user requirements. From preprocessing signature images to extracting features, fusing feature representations, and classifying signatures, activities delineate the key stages of the signature verification process, offering a structured view of the system's workflow.

Embedded within activities are actions, representing the atomic steps or operations performed as part of each activity. For instance, within the preprocessing activity, actions may include operations such as noise removal, resizing, and contrast enhancement, each contributing to preparing the signature images for subsequent feature extraction and analysis.

Transitions form the backbone of the activity diagram, delineating the flow of control between activities and actions based on certain conditions or events. Transitions define the sequence in which activities and actions are executed, guiding the progression of the system from one state to another. Guard conditions associated with transitions specify the criteria that must be met for a transition to occur, enabling dynamic control over the flow of activities based on the system's state and external inputs.

4.2.4 Sequence diagram:

A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".

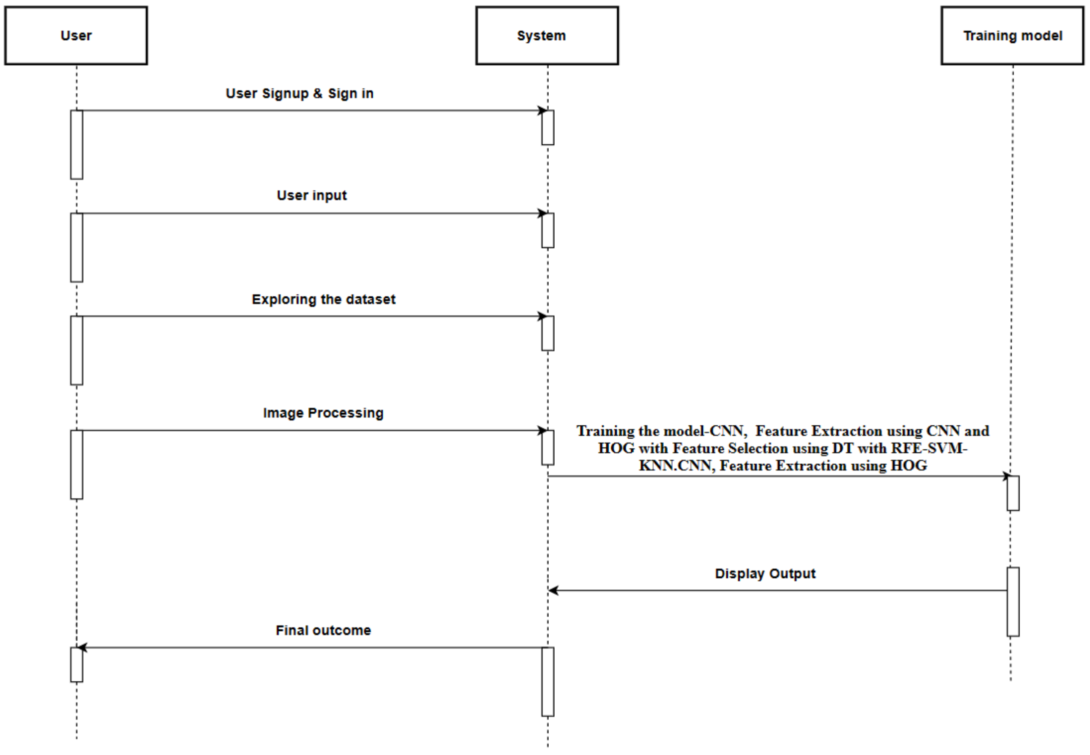

User
System
Training model
User Signup & Sign in
User input
Exploring the dataset
Image Processing
Training the model-CNN, Feature Extraction using CNN and HOG with Feature Selection using DT with RFE-SVM-KNN.CNN, Feature Extraction using HOG
Display Output
Final outcome

Figure 7: Sequence Diagram

4.2.5 Deployment diagram:

The deployment diagram captures the configuration of the runtime elements of the application. This diagram is by far most useful when a system is built and ready to be deployed.

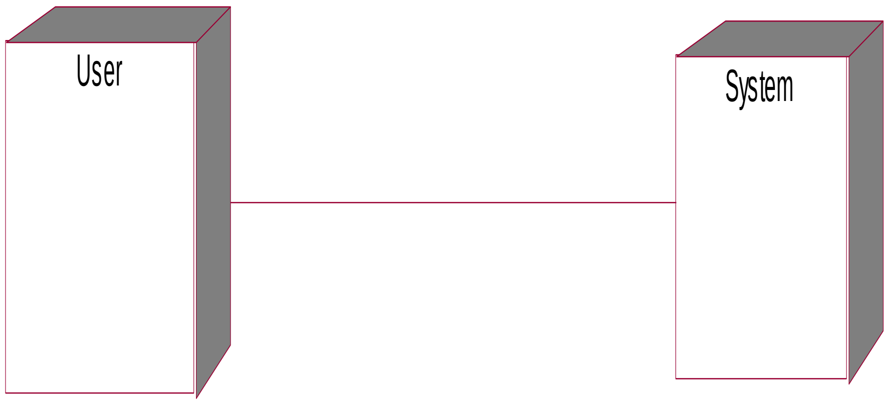


Figure 8: Deployment Diagram

The deployment diagram serves as a blueprint for visualizing the configuration of the runtime

elements of an application, offering a comprehensive overview of how software components and hardware resources are distributed and interconnected within the deployment environment. Unlike other diagrams that focus on the structural or behavioral aspects of the system, the deployment diagram provides insights into the physical deployment of system components, making it particularly valuable during the later stages of development when the system is ready to be deployed in a real-world environment.

The deployment diagram illustrates the allocation of software artifacts, such as executable files, libraries, and configuration files, to the various hardware nodes or computing devices within the deployment environment. Each software artifact is depicted as a deployment node, representing a logical unit of deployment that encapsulates one or more software components or modules. These deployment nodes may correspond to physical servers, virtual machines, containers, or other computing resources that host the software artifacts during runtime.

5. MODULES

1. Importing the packages

2. Exploring the dataset

    - Signature Dataset

    - UTsign

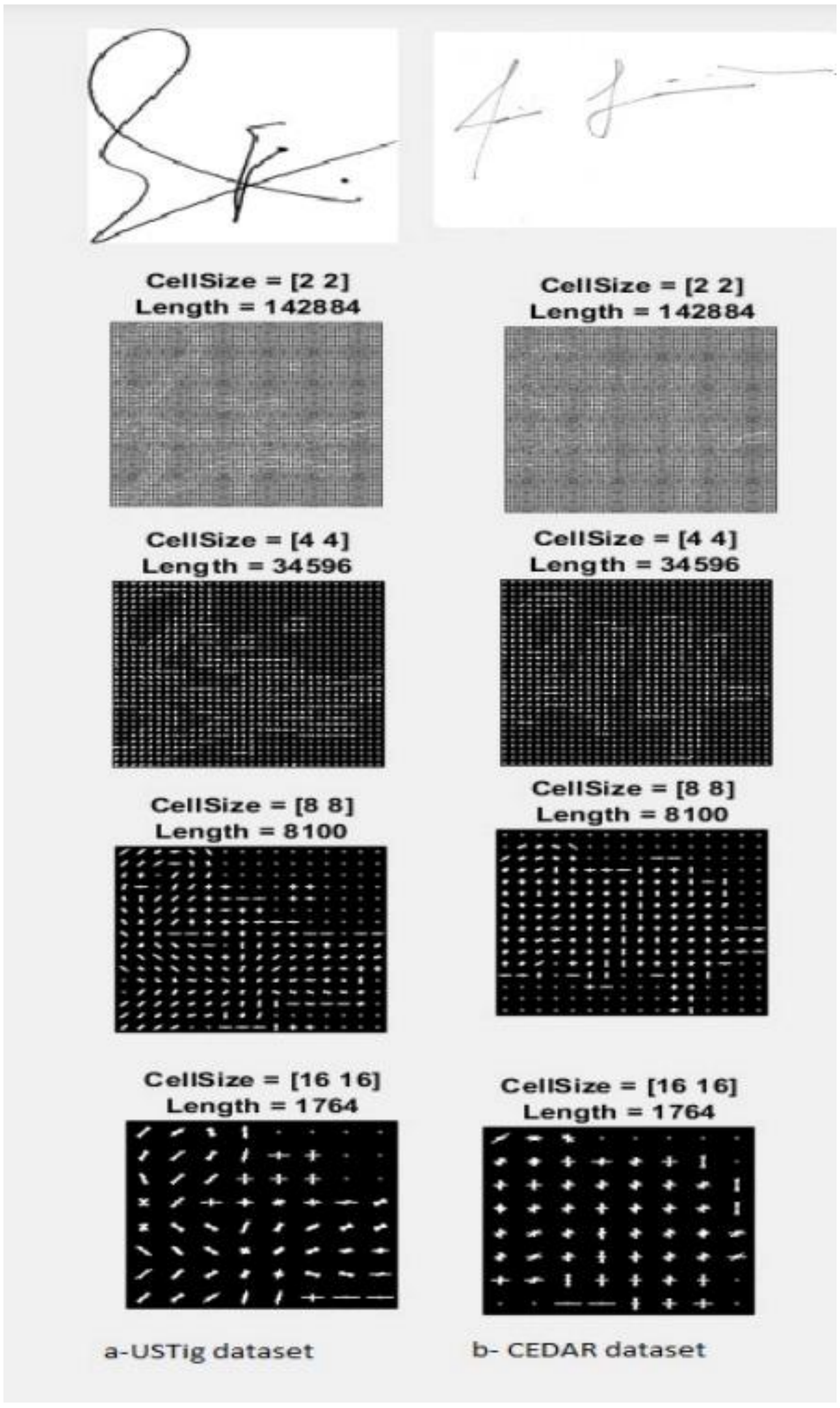
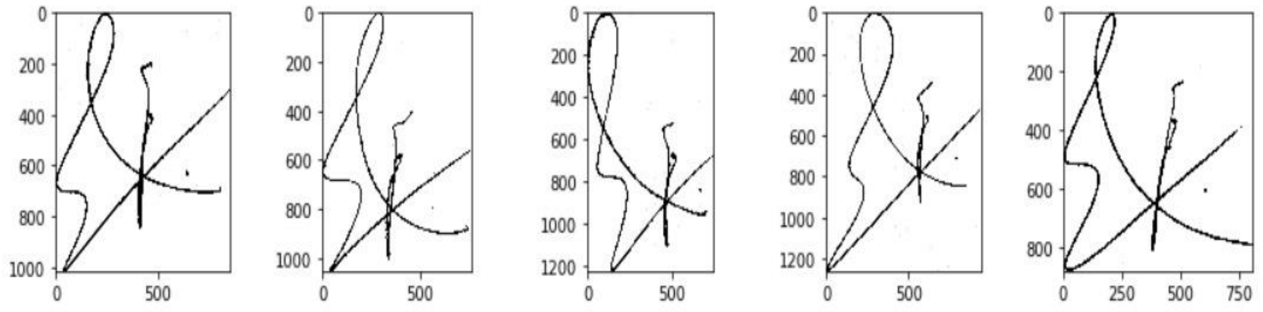


Figure 9: Signature Dataset (USTing dataset & CEDAR dataset)



3. Image processing
    - using Image Data Generator
        - Re-scaling the Image
        - Shear Transformation
        - Zooming the Image
        - Horizontal Flip
        - Reshaping the Image
    - Feature Extraction using CNN and HOG model
        - Reading the image
        - Resizing the image
        - Convert the Color
        - Appending the image and labels
        - Conversion to numpy value
        - Label Encoding
4. Building the model
    - CNN
        - Feature Extraction using CNN and HOG with Feature Selection using DT with RFE
            - SVM
            - KNN
            - LSTM
            - Voting Classifier (RF + DT)

- Xception
    - Feature Extraction
        - SVM
        - KNN
        - LSTM
        - Voting Classifier (RF + DT)

- Feature Extraction using HOG
    - SVM
    - KNN
    - LSTM
    - Voting Classifier (RF + DT)

5. Training the model

6. Building the model

Flask Framework

----

7. Flask Framework with Sqlite for signup and signin

8. Importing the packages

9. User Upload an image for analysis

10. The given input is preprocessed

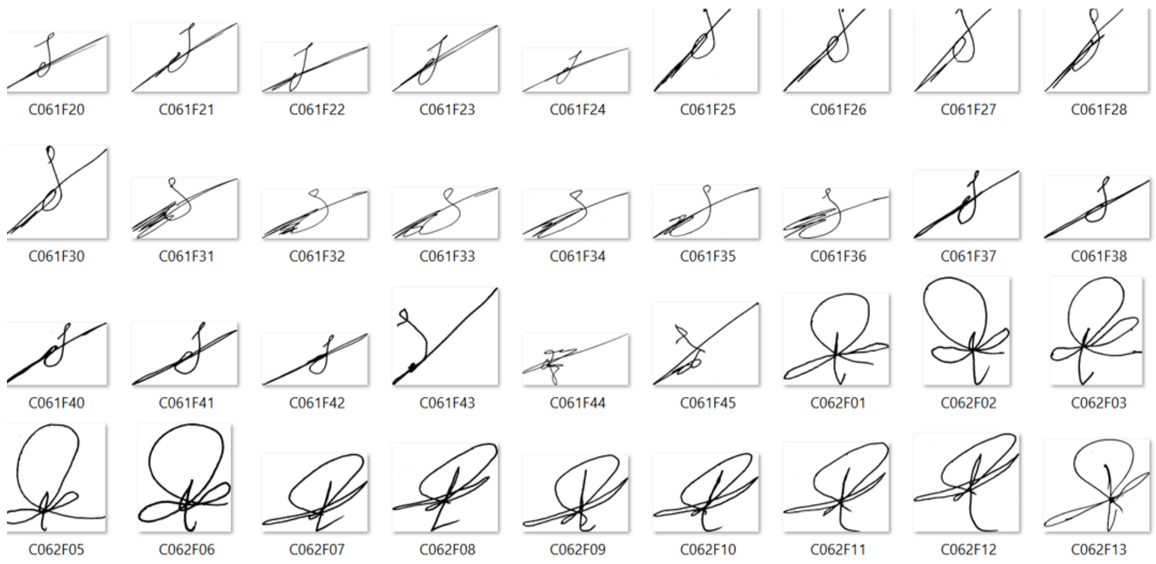


Figure 10: Image preprocessing

11. The trained model is used for predicting the result
12. Final Outcome is display

6. IMPLEMENTATION

- User signup & login: This module will get registration and login

1. Data exploration: This module we will load data (Train data, Test data) into system after User signup & login, we will get registration and login.
2. Data Processing: This module we will read data for processing (Splitting data into train & test)
3. Model generation: Model building - CNN, Feature Extraction using HOG, Feature Extraction using CNN, HOG and combination if CNN & HOG
4. Prediction: final predicted displayed as forgery or genuine

It deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

Platform – In computing, a platform describes some sort of framework, either in hardware or software, which allows software to run. Typical platforms include a computer's architecture, operating system, or programming languages and their runtime libraries.

Operating system is one of the first requirements mentioned when defining system requirements (software). Software may not be compatible with different versions of same line of operating systems, although some measure of backward compatibility is often maintained. For example, most software designed for Microsoft Windows XP does not run on Microsoft Windows 98, although the converse is not always true. Similarly, software designed using newer features of Linux Kernel v2.6 generally does not run or compile properly (or at all) on Linux distributions using Kernel v2.2 or v2.4.

APIs and drivers – Software making extensive use of special hardware devices, like high-end display adapters, needs special API or newer device drivers. A good example is DirectX, which is a collection of APIs for handling tasks related to multimedia, especially game programming, on Microsoft platforms.

Web browser – Most web applications and software depending heavily on Internet technologies make use of the default browser installed on system. Microsoft Internet Explorer is a frequent choice of software running on Microsoft Windows, which makes use of ActiveX controls, despite their vulnerabilities.

Anaconda

Anaconda is a widely used open-source distribution platform for data science and machine learning workflows. It provides a comprehensive ecosystem of tools and packages that simplify the process of setting up and managing environments for data analysis, scientific computing, and AI development. Founded by Continuum Analytics, Anaconda aims to streamline the deployment of Python and R-based applications by offering a convenient package management system and pre-built environments.

Anaconda also offers a powerful environment management system through its conda package manager. Conda enables users to create isolated environments with specific configurations and dependencies, making it easy to maintain consistency and avoid conflicts between different projects. With conda, users can create environments with different versions of Python, as well as custom combinations of libraries and packages tailored to their specific needs. This flexibility is especially useful for collaborative projects or when working with legacy codebases that require specific dependencies.

Another notable aspect of Anaconda is its support for both Python and R programming languages. While Python is the dominant language in the data science community, Anaconda's integration with R allows users to leverage the extensive ecosystem of R packages for statistical analysis, data visualization, and machine learning. By providing a unified platform for both Python and R, Anaconda caters to a broader audience of data scientists, researchers, and developers with diverse backgrounds and preferences.

Anaconda also offers enterprise-grade solutions for organizations and teams working on data-intensive projects. Anaconda Enterprise provides advanced features such as centralized package management, role-based access control, and scalable infrastructure for deploying and managing data science workflows in production environments. With Anaconda Enterprise, organizations can streamline collaboration, ensure regulatory compliance, and accelerate the deployment of AI-driven applications across their enterprise.

In addition to its core distribution platform, Anaconda supports a thriving community of developers, data scientists, and educators through its online forums, documentation, and training resources. The Anaconda community actively contributes to the development of open-source projects, shares best practices, and provides support to users facing technical challenges or seeking advice on how to optimize their workflows.

Overall, Anaconda has emerged as a leading platform for data science and machine learning, offering a comprehensive suite of tools, libraries, and services that empower individuals and organizations to tackle complex data challenges with confidence. Whether you're a beginner exploring the fundamentals of data analysis or a seasoned professional deploying advanced AI models in production, Anaconda provides the tools and resources you need to succeed in the rapidly evolving field of data science.

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

Architecture – All computer operating systems are designed for a particular computer architecture. Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture. See also a list of common operating systems and their supporting architectures.

Processing power – The power of the central processing unit (CPU) is a fundamental system requirement for any software. Most software running on x86 architecture define processing power as the model and the clock speed of the CPU. Many other features of a CPU that influence its speed and power, like bus speed, cache, and MIPS are often ignored. This definition of power is often erroneous, as AMD Athlon and Intel Pentium CPUs at similar clock speed often have different throughput speeds. Intel Pentium CPUs have enjoyed a considerable degree of popularity, and are often mentioned in this category.

Memory – All software, when run, resides in the random access memory (RAM) of a computer. Memory requirements are defined after considering demands of the application, operating system, supporting software and files, and other running processes. Optimal performance of other unrelated software running on a multi-tasking computer system is also considered when defining this requirement.

Secondary storage – Hard-disk requirements vary, depending on the size of software installation, temporary files created and maintained while installing or running the software, and possible use of swap space (if RAM is insufficient).

Display adapter – Software requiring a better than average computer graphics display, like graphics editors and high-end games, often define high-end display adapters in the system requirements.

Peripherals – Some software applications need to make extensive and/or special use of some peripherals, demanding the higher performance or functionality of such peripherals. Such peripherals include CD-ROM drives, keyboards, pointing devices, network devices, etc.

Note: As an extension we applied an CNN model and CNN with HOG feature , HOG based feature extraction method, along with they mention to use SVM, KNN and LSTM model for analysis the feature extracted data and got 95,95.5 and 91.3% of accuracy,

As an extension we have applied Voting Classifier for analysis Dataset, in which got 100% of accuracy based Feature Extraction using Voting Classifier.

Algorithms:

CNN: A Convolutional Neural Network (CNN) is a deep learning architecture designed for image and spatial data analysis. It utilizes convolutional layers to automatically learn and extract hierarchical features from input data, enabling tasks like image recognition, classification, and object detection with high accuracy by capturing patterns and spatial relationships in the data.

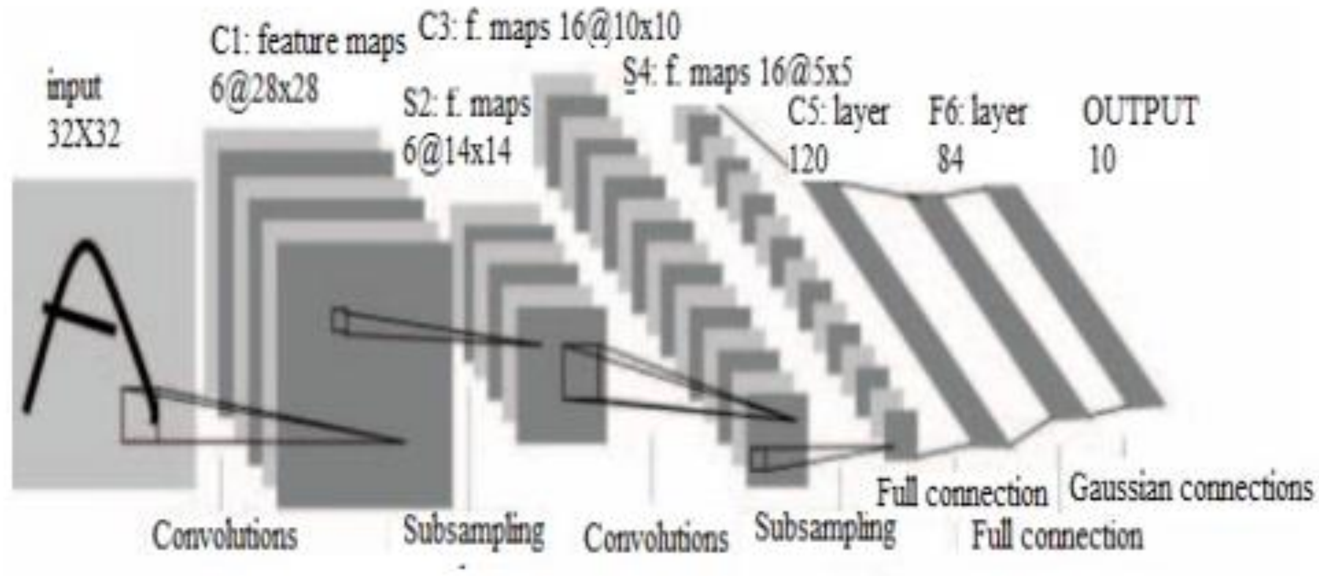


Figure 11: Convolutional Neural Network (CNN)

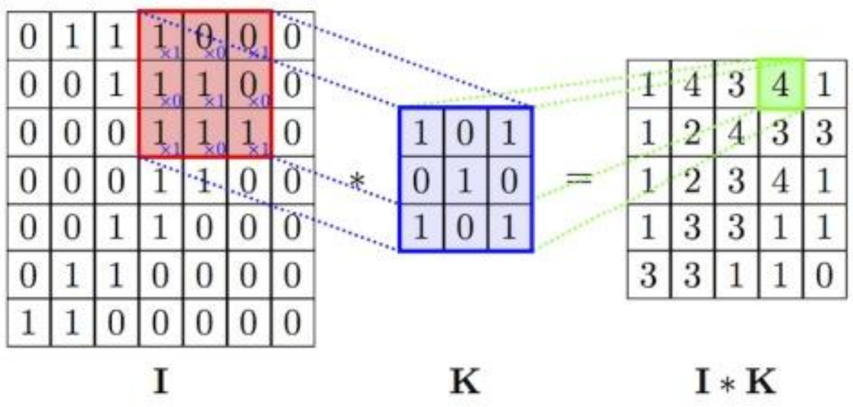


Figure 12: CNN using KNN

Feature Extraction using HOG: Feature extraction using Histogram of Oriented Gradients (HOG) is a computer vision technique that quantifies local image gradient orientations to represent and capture the shape and texture information within an image. It is commonly used for object detection and recognition, providing a concise representation of visual features for various applications, particularly in image processing and computer vision tasks.

Feature Extraction using CNN and HOG with Feature Selection using DT with RFE: Feature extraction combines Convolutional Neural Network (CNN) and Histogram of Oriented Gradients (HOG) techniques to capture rich image features. A Decision Tree-based Recursive Feature Elimination (RFE) method is then employed for feature selection. This hybrid approach enhances pattern recognition and reduces dimensionality, improving the efficiency and effectiveness of image-based tasks such as object recognition and signature verification.

SVM: Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It works by finding a hyperplane that maximally separates data points into distinct classes in a high-dimensional space. SVM is effective for complex data with non-linear boundaries, thanks to kernel functions that map data into higher dimensions for improved separation.

SVM aims to find the optimal hyperplane that best separates the data points of different classes in the feature space. The hyperplane is defined as the decision boundary that maximizes the margin, which is the distance between the hyperplane and the nearest data points from each class, known as support vectors. The intuition behind SVM is to find a hyperplane that not only separates the data but also maximizes the margin, making it robust to new unseen data.

KNN: K-Nearest Neighbors (KNN) is a simple supervised machine learning algorithm used for classification and regression tasks. It determines the class or value of a data point by examining the majority class or average value among its 'k' nearest neighboring data points in a feature space, making it particularly intuitive for pattern recognition and similarity-based tasks.

KNN relies on the principle of similarity, where the class or value of a data point is determined by the majority vote or averaging of its nearest neighbors in the feature space. The algorithm operates on the assumption that similar data points tend to belong to the same class or have similar output values. In the case of classification, KNN assigns the most common class label among its k nearest neighbors, where k is a user-defined parameter. For regression tasks, KNN calculates the average or weighted average of the output values of its k nearest neighbors to predict the value of the target variable. The choice of k is crucial in KNN, as it determines the balance between bias and variance in the model. A smaller value of k leads to a more flexible model with higher variance but lower bias, while a larger value of k results in a smoother decision boundary or regression curve with lower variance but higher bias.

KNN does not involve an explicit training phase; instead, it stores all the training data points and performs predictions at runtime by computing distances (e.g., Euclidean distance or Manhattan distance) between the query data point and the training data points. While KNN is simple to implement and easy to understand, it has limitations, such as computational inefficiency, especially with large datasets, and sensitivity to the choice of distance metric and the number of neighbors (k). Moreover, KNN's performance can degrade significantly in high-dimensional spaces due to the curse of dimensionality, where the density of data points becomes sparse, making it challenging to identify meaningful nearest neighbors. Despite its drawbacks, KNN remains a popular choice for small to medium-sized datasets, especially in domains where interpretability and simplicity are prioritized over computational efficiency, such as healthcare, recommendation systems, and anomaly detection.

LSTM: Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture used in deep learning. It is designed to model sequential data and has the ability to capture long-term dependencies and remember information over extended sequences. LSTMs are commonly used in natural language processing and time series analysis tasks.

LSTMs are widely used in various tasks such as natural language processing, time series prediction, and speech recognition, where sequential data is prevalent. The key innovation of LSTM lies in its ability to selectively retain or forget information over long time periods through gated units, which consist of a memory cell, input gate, forget gate, and output gate. The memory cell serves as a persistent memory unit that stores information over time, while the gates regulate the flow of information into and out of the memory cell.

During the forward pass of training or inference, LSTM processes input sequences step by step, updating the state of its memory cell and gates at each time step. The input gate controls the extent to which new information is incorporated into the memory cell, while the forget gate determines which information from the previous time step should be discarded. The output gate governs the amount of information that is propagated to the next time step and outputted to the external environment. This gating mechanism enables LSTMs to effectively capture long-range dependencies and mitigate the vanishing gradient problem, which is common in traditional RNNs and hinders learning in sequences with long-term dependencies.

Additionally, LSTMs can be stacked to form deep LSTM architectures, where multiple LSTM layers are connected sequentially. Deep LSTM networks can capture hierarchical representations of sequential data and learn complex patterns across multiple abstraction levels. Moreover, LSTMs can be bidirectional, allowing information to flow both forward and backward through time. Bidirectional LSTMs combine information from past and future time steps, enabling the model to make more informed predictions based on the entire context of the input sequence.

Training LSTMs involves optimizing the network parameters (e.g., weights and biases) using backpropagation through time (BPTT) or other variants of gradient descent. The loss function is typically defined based on the task at hand, such as cross-entropy loss for classification or mean squared error for regression. During training, the network learns to update its parameters to minimize the discrepancy between the predicted output and the ground truth labels or targets.

In summary, LSTM is a powerful architecture for modeling sequential data, offering advantages such as the ability to capture long-term dependencies, mitigate vanishing gradients, and handle variable-length sequences. Its gated structure enables effective memory management and learning of complex temporal patterns. Despite its computational complexity and training requirements, LSTM has become a fundamental building block in many state-of-the-art models for sequential data processing and has contributed to significant advancements in various fields such as natural language understanding, speech recognition, and time series forecasting.

Voting Classifier (RF + DT): A Voting Classifier combines the predictions of multiple machine learning models to make a final decision. In this case, it combines the predictions of a Random Forest (RF) and a Decision Tree (DT) classifier. The Voting Classifier typically selects the class predicted by the majority of the constituent models, enhancing overall prediction accuracy and robustness.

SAMPLE CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import cv2
import glob
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import applications
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import Sequential
from tensorflow.keras.models import Sequential, Model, load_model
from tensorflow.keras import applications
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Dropout, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping,
ReduceLROnPlateau, TensorBoard
from tensorflow.keras import backend as K
from tensorflow.keras.preprocessing.image import img_to_array
import gc
from tensorflow.keras.models import Model
from PIL import Image
import pickle
train_dir="../input/signature-verification-dataset-iraninan/train"
test_dir="../input/signature-verification-dataset-iraninan/test"
img = plt.imread('../input/signature-verification-dataset-iraninan/train/04/04-01.jpg')
plt.imshow(img)
img = plt.imread('../input/signature-verification-dataset-iraninan/train/04-f/04-f-01.jpg')
plt.imshow(img)
train_data_names = []
```

```
test_data_names = []

train_data = []
train_labels = []

for per in os.listdir('../input/signature-verification-dataset-iraninan/train'):
    for data in glob.glob('../input/signature-verification-dataset-iraninan/train/'+per+'/*.*'):

        train_data_names.append(data)
        img = cv2.imread(data)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (SIZE,SIZE))
        train_data.append([img])
        if per[-1]=='f':
            train_labels.append(np.array(1))
        else:
            train_labels.append(np.array(0))

train_data = np.array(train_data)/255.0
train_labels = np.array(train_labels)

#Test Data

test_data = []
test_labels = []

for per in os.listdir('../input/signature-verification-dataset-iraninan/test'):
    for data in glob.glob('../input/signature-verification-dataset-iraninan/test/'+per+'/*.*'):
        test_data_names.append(data)
        img = cv2.imread(data)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (SIZE,SIZE))
        test_data.append([img])
        if per[-1]=='f':
            test_labels.append(np.array(1))
        else:
            test_labels.append(np.array(0))

test_data = np.array(test_data)/255.0
test_labels = np.array(test_labels)
train_labels[1:1000]
with open('./train_data_names.pkl', 'wb') as fp:
    pickle.dump(train_data_names, fp)

with open('./test_data_names.pkl', 'wb') as fp:
    pickle.dump(test_data_names, fp)
# Categorical labels
print(train_labels.shape)
train_labels = to_categorical(train_labels)
print(train_data.shape)
# Reshaping
train_data = train_data.reshape(-1, SIZE,SIZE, 3)
test_data = test_data.reshape(-1, SIZE,SIZE, 3)
print(train_data.shape)
print(test_data.shape)
base_model = applications.VGG16(weights='imagenet', include_top=False, input_shape=input_)

model = Sequential()
data_augmentation = keras.Sequential([layers.experimental.preprocessing.RandomRotation(0.1)])
model.add(base_model)
model.add(Flatten(input_shape=base_model.output_shape[1:]))
model.add(Dense(256, activation='relu'))
model.add(Dense(output_, activation='sigmoid'))

model = Model(inputs=model.input, outputs=model.output)
model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(lr=1e-4),
              metrics=['accuracy'])

model.summary()
```

```
earlyStopping = EarlyStopping(monitor='val_loss',
                    min_delta=0,
                    patience=3,
                    verbose=1)

early_stop=[earlyStopping]
progess = model.fit(train_data,train_labels, batch_size=BS,epochs=EPOCHS,
callbacks=early_stop,validation_split=.3)
acc = progess.history['accuracy']
val_acc = progess.history['val_accuracy']
loss = progess.history['loss']
val_loss = progess.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.show()

plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()

intermediate_layer_model = Model(inputs=model.input,
                     outputs=model.layers[-2].output)
intermediate_output_train = intermediate_layer_model.predict(train_data)
intermediate_output_test = intermediate_layer_model.predict(test_data)

np.save('./VGG16_Adam_train', intermediate_output_train)
np.save('./VGG16_Adam_test', intermediate_output_test)
```

```
model.save('kerasVggSigFeatures.h5')
base_model = applications.InceptionV3(weights='imagenet', include_top=False, input_shape=input_)

model = Sequential()
data_augmentation = keras.Sequential([layers.experimental.preprocessing.RandomRotation(0.1)])
model.add(base_model)
model.add(Flatten(input_shape=base_model.output_shape[1:]))
model.add(Dense(256, activation='relu'))
model.add(Dense(output_, activation='softmax'))

model = Model(inputs=model.input, outputs=model.output)
model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizers.Adam(lr=1e-4),
        metrics=['accuracy'])

model.summary()

earlyStopping = EarlyStopping(monitor='val_loss',
                    min_delta=0,
                    patience=3,
                    verbose=1)

early_stop=[earlyStopping]
train_labels = np.argmax(train_labels, axis=1)
progess = model.fit(train_data,train_labels, batch_size=BS,epochs=EPOCHS,
callbacks=early_stop,validation_split=.3)
acc = progess.history['accuracy']
val_acc = progess.history['val_accuracy']
loss = progess.history['loss']
val_loss = progess.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.show()

plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()

intermediate_layer_model = Model(inputs=model.input,
                                 outputs=model.layers[-2].output)
intermediate_output_train = intermediate_layer_model.predict(train_data)
intermediate_output_test = intermediate_layer_model.predict(test_data)

np.save('./InceptionV3_Adam_train', intermediate_output_train)
np.save('./InceptionV3_Adam_test', intermediate_output_test)
base_model = applications.InceptionV3(weights='imagenet', include_top=False, input_shape=input_)

model = Sequential()
data_augmentation = keras.Sequential([layers.experimental.preprocessing.RandomRotation(0.1)])
model.add(base_model)
model.add(Flatten(input_shape=base_model.output_shape[1:]))
model.add(Dense(256, activation='relu'))
model.add(Dense(output_, activation='softmax'))

model = Model(inputs=model.input, outputs=model.output)
model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizers.Adagrad(lr=1e-4),
              metrics=['accuracy'])

model.summary()

earlyStopping = EarlyStopping(monitor='val_loss',
                              min_delta=0,
                              patience=3,
                        verbose=1)

early_stop=[earlyStopping]
progess = model.fit(train_data,train_labels, batch_size=BS,epochs=EPOCHS,
callbacks=early_stop,validation_split=.3)
acc = progess.history['accuracy']
val_acc = progess.history['val_accuracy']
loss = progess.history['loss']
val_loss = progess.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.show()

plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()

intermediate_layer_model = Model(inputs=model.input,
                                 outputs=model.layers[-2].output)
intermediate_output_train = intermediate_layer_model.predict(train_data)
intermediate_output_test = intermediate_layer_model.predict(test_data)

np.save('./InceptionV3_Adagrad_train', intermediate_output_train)
np.save('./InceptionV3_Adagrad_test', intermediate_output_test)
import numpy as np
import pickle
import pandas as pd
```

```
from sklearn.metrics import accuracy_score
from tqdm import tqdm_notebook as tqdm
from sklearn.metrics import confusion_matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import random
from scipy.spatial import distance
from sklearn import svm
from sklearn.metrics import precision_recall_fscore_support
import warnings
warnings.filterwarnings('ignore')
def dataExt(model, optimizer):
    # Load feature ext Data
    filesTrain = "/kaggle/working/train_data_names.pkl"
    filesTest = "/kaggle/working/test_data_names.pkl"
    #../input/feature-extracted/InceptionV3_features/InceptionV3_Adagrad_test.npy
    pathTrain = "/kaggle/working/VGG16_Adam_train.npy"
    pathTest = "/kaggle/working/VGG16_Adam_test.npy"

    # unload pickle the file names
    with open(filesTrain,'rb') as f:
        file_train_list = np.load(f, allow_pickle=True)

    with open(filesTest,'rb') as f:
        file_test_list = np.load(f, allow_pickle=True)
    # data preprocessing
    file_train_list = [i[55:] for i in file_train_list]
    file_test_list = [i[54:] for i in file_test_list]

    feat_train_np = np.load(pathTrain)
    feat_test_np = np.load(pathTest)
    # return all the data of features of specific model
    return file_train_list, file_test_list, feat_train_np, feat_test_np
```

```
def PolySVM(model, optimizer, X_train, y_train, X_test, y_test):
    print("POLY SVM", model, optimizer)

    a,b,c,d = dataExt(model, optimizer)

    def name2feat(string):
        try:
            index = a.index(string)
            return c[index]
        except:
            index = b.index(string)
            return d[index]
    X_train['img1'] = X_train['img1'].apply(name2feat)
    X_train['img2'] = X_train['img2'].apply(name2feat)
    X_test['img1'] = X_test['img1'].apply(name2feat)
    X_test['img2'] = X_test['img2'].apply(name2feat)

    new_data_train = []
    for index,row in X_train.iterrows():
        new_list = list(row[0])
        new_list.extend(row[1])
        new_data_train.append(new_list)

    new_data_test = []
    for index,row in X_test.iterrows():
        new_list = list(row[0])
        new_list.extend(row[1])
        new_data_test.append(new_list)

    Model = svm.SVC(kernel='poly') #rbf by default svm.SVC()
    Model.fit(new_data_train, y_train)
    with open('svm-'+model+'-'+optimizer+'.pkl','wb') as f:
        pickle.dump(Model,f)
    y_pred = Model.predict(new_data_test)
    print("Acuracy", accuracy_score(y_test, y_pred))
    print("P,R,F1:",precision_recall_fscore_support(y_test, y_pred, average='macro'))
    df_cm = pd.DataFrame(confusion_matrix(y_test,y_pred,normalize = 'true'), index = [i for i in "01"],
                columns = [i for i in "01"])
    plt.figure(figsize = (10,7))
    sn.heatmap(df_cm, annot=True)

import os
import csv

def find_files_in_folder(path, folder_name):
    folder_path = None
    for root, dirs, files in os.walk(path):
        if folder_name in dirs:
            folder_path = os.path.join(root, folder_name)
            break
    img1 = []
    if folder_path:
        files = os.listdir(folder_path)
        for file in files:
            folder_name = os.path.dirname(file)
            file_name = os.path.basename(file)
            img1.append(file_name)
        return img1
    else:
        return None

# Example usage
folder_train_path = '../input/signature-verification-dataset-iraninan/train'  # Replace with the actual folder path
folder_test_path = '../input/signature-verification-dataset-iraninan/test'  # Replace with the actual folder path

# Get all files in the folder
files_train = os.listdir(folder_train_path)
```

```
files_test = os.listdir(folder_test_path)
img1_train = []
img2_train = []
img1_test = []
img2_test = []

for file in files_train:
    files_in_folder = find_files_in_folder(folder_train_path, file)
    if "-f" in file:
        img1_train.append(file)
    else:
        img2_train.append(file)

for file in files_test:
    files_in_folder = find_files_in_folder(folder_test_path, file)
    if "-f" in file:
        img1_test.append(file)
    else:
        img2_test.append(file)

# Write folder name and file name to a CSV file
with open('train_data.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['img1', 'img2', 'target'])
    for real in sorted(img2_train):
        files_in_real_folder = find_files_in_folder(folder_train_path, real)
        for real_file in files_in_real_folder:
            for forg in sorted(img1_train):
                f = forg.split('-f')
                if f[0] == real:
                    files_in_forg_folder = find_files_in_folder(folder_train_path, forg)
                    for forg_file in files_in_forg_folder:
                        writer.writerow([real+'/'+real_file, forg+'/'+forg_file, 1])
                    break
            for real_file_2 in files_in_real_folder:
                if real_file != real_file_2:
                    writer.writerow([real+'/'+real_file, real+'/'+real_file_2, 0])

# Write folder name and file name to a CSV file
with open('test_data.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['img1', 'img2', 'target'])
    for real in sorted(img2_test):
        files_in_real_folder = find_files_in_folder(folder_test_path, real)
        for real_file in files_in_real_folder:
            for forg in sorted(img1_test):
                f = forg.split('-f')
                if f[0] == real:
                    files_in_forg_folder = find_files_in_folder(folder_test_path, forg)
                    for forg_file in files_in_forg_folder:
                        writer.writerow([real+'/'+real_file, forg+'/'+forg_file, 1])
                    break
            for real_file_2 in files_in_real_folder:
                if real_file != real_file_2:
                    writer.writerow([real+'/'+real_file, real+'/'+real_file_2, 0])

# Write folder name and file name to a CSV file
with open('data.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['img1', 'img2', 'target'])
    for real in sorted(img2_train):
        files_in_real_folder = find_files_in_folder(folder_train_path, real)
        for real_file in files_in_real_folder:
            for real2 in sorted(img2_train):
                files_in_real_folder2 = find_files_in_folder(folder_train_path, real2)
                for real_file2 in files_in_real_folder2:
                    if real_file != real_file2 and int(real) < 32 and int(real2) < 32:
                        if real == real2:
                            target = 0
                        else:
```

```
                    target = 1
                writer.writerow([real+"/"+real_file, real2+'/'+real_file2, target])
%%time
data = pd.read_csv('/kaggle/working/train_data.csv')
X_data = data[['img1','img2']]
y_data = data[['target']]
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.33)
PolySVM('VGG16','RMSprop', X_train, y_train, X_test, y_test)
%%time
data = pd.read_csv('/kaggle/working/train_data.csv')
X_data = data[['img1','img2']]
y_data = data[['target']]
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.33)
PolySVM('InceptionV3','Adam', X_train, y_train, X_test, y_test)
%%time
data = pd.read_csv('/kaggle/working/train_data.csv')
X_data = data[['img1','img2']]
y_data = data[['target']]
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.33)
LogReg('InceptionV3','Adagrad', X_train, y_train, X_test, y_test)
```

7. SOFTWARE ENVIRONMENT

MACHINE LEARNING:

Before we take a look at the details of various machine learning methods, let's start by looking at what machine learning is, and what it isn't. Machine learning is often categorized as a subfield of artificial intelligence, but I find that categorization can often be misleading at first brush. The study of machine learning certainly arose from research in this context, but in the data science application of machine learning methods, it's more helpful to think of machine learning as a means of *building models of data*.

Fundamentally, machine learning involves building mathematical models to help understand data. "Learning" enters the fray when we give these models *tunable parameters* that can be adapted to observed data; in this way the program can be considered to be "learning" from the data. Once these models have been fit to previously seen data, they can be used to predict and understand aspects of newly observed data. I'll leave to the reader the more philosophical digression regarding the extent to which this type of mathematical, model-based "learning" is similar to the "learning" exhibited by the human brain. Understanding the problem setting in machine learning is essential to using these tools effectively, and so we will start with some broad categorizations of the types of approaches we'll discuss here.

Challenges in Machines Learning: -

While Machine Learning is rapidly evolving, making significant strides with cybersecurity and autonomous cars, this segment of AI as whole still has a long way to go. The reason behind is that ML has not been able to overcome number of challenges. The challenges that ML is facing currently are −

Quality of data − Having good-quality data for ML algorithms is one of the biggest challenges. Use of low-quality data leads to the problems related to data preprocessing and feature extraction.

Time-Consuming task − Another challenge faced by ML models is the consumption of time especially for data acquisition, feature extraction and retrieval.

Lack of specialist persons − As ML technology is still in its infancy stage, availability of expert resources is a tough job.

No clear objective for formulating business problems − Having no clear objective and well-defined goal for business problems is another key challenge for ML because this technology is not that mature yet.

Issue of over fitting & under fitting − If the model is over fitting or under fitting, it cannot be represented well for the problem.

Curse of dimensionality − another challenge ML model faces is too many features of data points. This can be a real hindrance.

Difficulty in deployment − Complexity of the ML model makes it quite difficult to be deployed in real life.

DEEP LEARNING

Deep learning is a branch of machine learning which is based on artificial neural networks. It is capable of learning complex patterns and relationships within data. In deep learning, we don't need to explicitly program everything. It has become increasingly popular in recent years due to the advances in processing power and the availability of large datasets. Because it is based on artificial neural networks (ANNs) also known as deep neural networks (DNNs). These neural networks are inspired by the structure and function of the human brain's biological neurons, and they are designed to learn from large amounts of data.

What is Anaconda for Python?

Anaconda Python is a free, open-source platform that allows you to write and execute code in the programming language Python. It is by continuum.io, a company that specializes in Python development. The Anaconda platform is the most popular way to learn and use Python for scientific computing, data science, and machine learning. It is used by over thirty million people worldwide and is available for Windows, macOS, and Linux.

People like using Anaconda Python because it simplifies package deployment and management. It also comes with a large number of libraries/packages that you can use for your projects. Since Anaconda Python is free and open-source, anyone can contribute to its development.

What is Anaconda for Python?

Anaconda software helps you create an environment for many different versions of Python and package versions. Anaconda is also used to install, remove, and upgrade packages in your project environments. Furthermore, you may use Anaconda to deploy any required project with a few mouse clicks. This is why it is perfect for beginners who want to learn Python.

Now that you know what Anaconda Python is, let's look at how to install it.

How to install Anaconda for Python?

To install Anaconda, just head to the Anaconda Documentation website and follow the instructions to download the installer for your operating system. Once the installer successfully downloads, double-click on it to start the installation process.

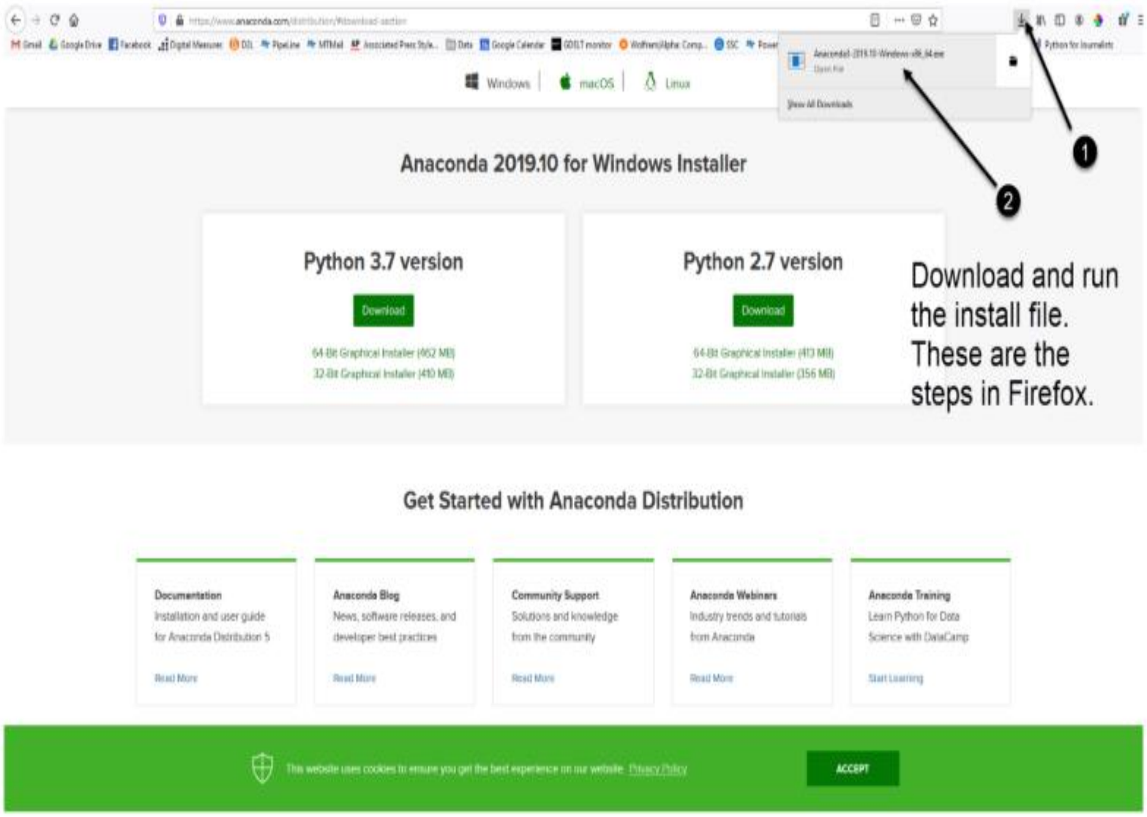
Follow the prompts and agree to the terms and conditions. When you are asked if you want to "add Anaconda to my PATH environment variable," make sure that you select "yes." This will ensure that Anaconda is added to your system's PATH, which is a list of directories that your operating system uses to find the files it needs.

Once the installation is complete, you will be asked if you want to "enable Anaconda as my default Python." We recommend selecting "yes" to use Anaconda as your default Python interpreter.

Python Anaconda Installation

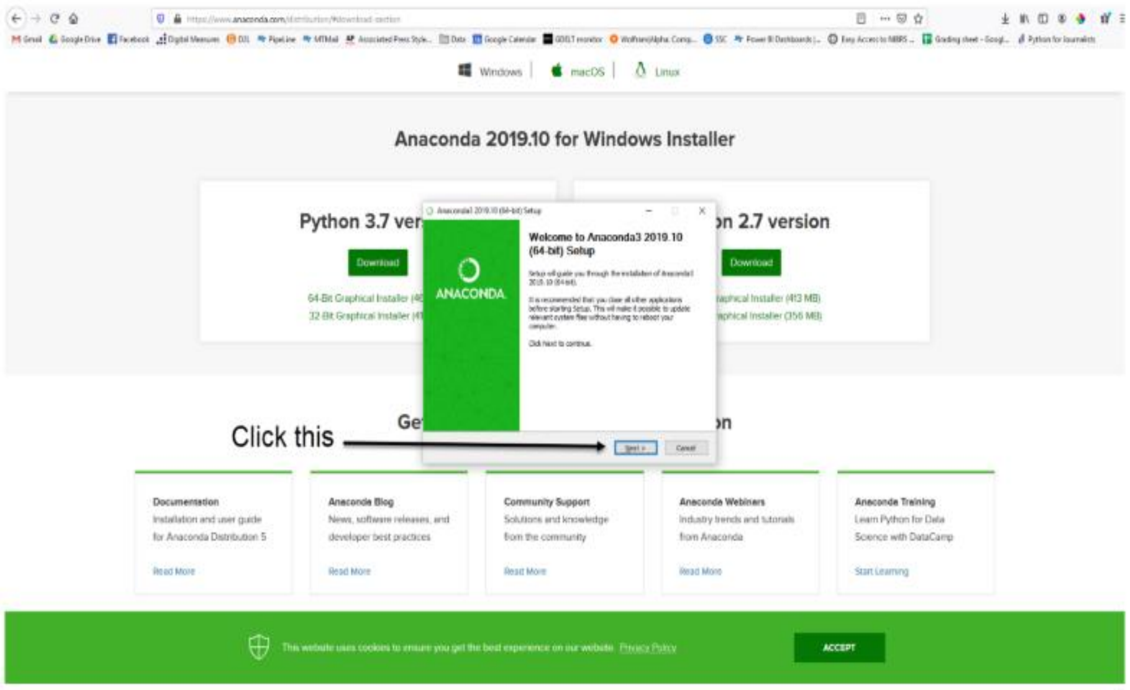
Next in the Python anaconda tutorial is its installation. The latest version of Anaconda at the time of writing is 2019.10. Follow these steps to download and install Anaconda on your machine:

1. Go to this link and download Anaconda for Windows, Mac, or Linux: – Download anaconda

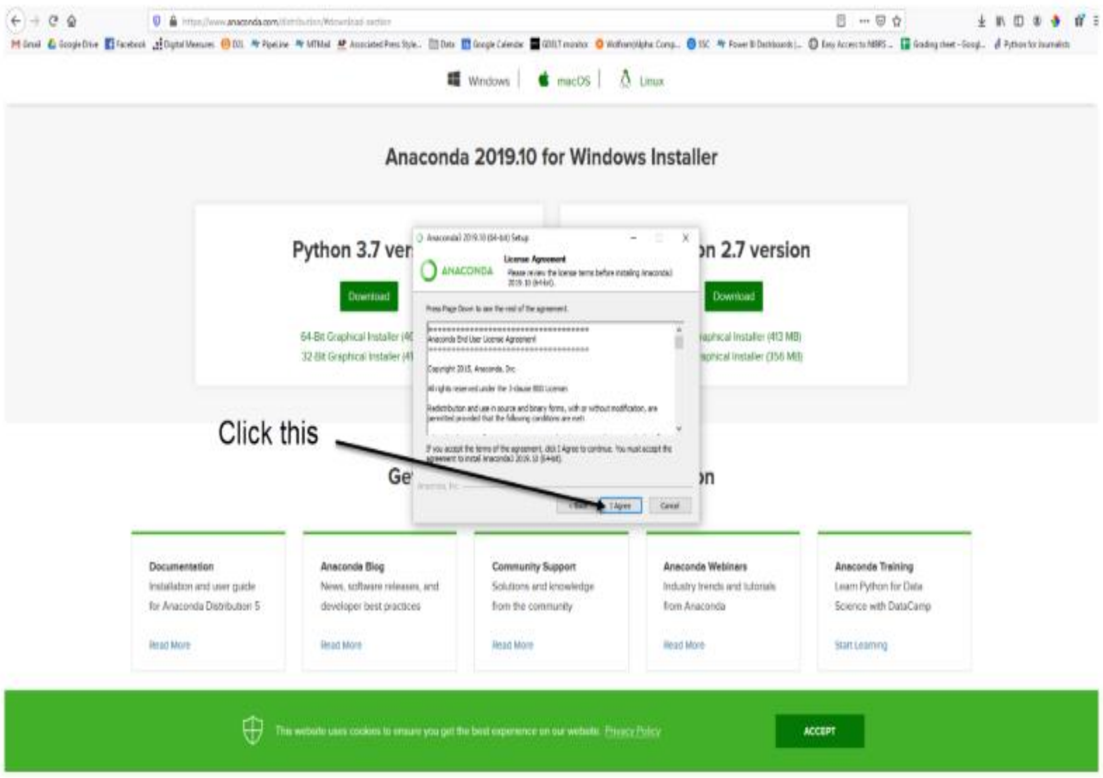

Windows
macOS
Linux
Anaconda 2019.10 for Windows Installer
Python 3.7 version
Download
64-Bit Graphical Installer (462 MB)
32-Bit Graphical Installer (410 MB)
Python 2.7 version
Download
64-Bit Graphical Installer (413 MB)
32-Bit Graphical Installer (356 MB)
1
2
Download and run the install file. These are the steps in Firefox.
Get Started with Anaconda Distribution
Documentation
Anaconda Blog
Community Support
Anaconda Webinars
Anaconda Training
ACCEPT

You can download the installer for Python 3.7 or for Python 2.7 (at the time of writing). And you can download it for a 32-bit or 64-bit machine.

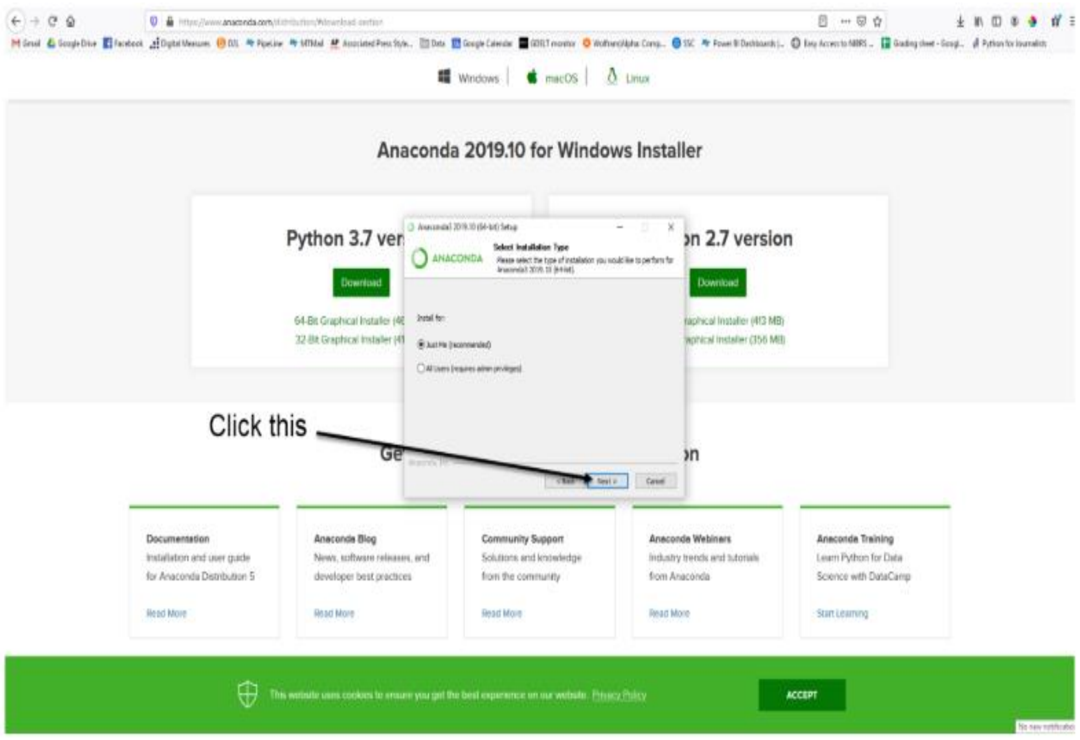
2. Click on the downloaded .exe to open it. This is the Anaconda setup. Click next.


Anaconda 2019.10 for Windows Installer
Welcome to Anaconda3 2019.10 (64-bit) Setup
ANACONDA
Click this
Next >
Cancel
ACCEPT

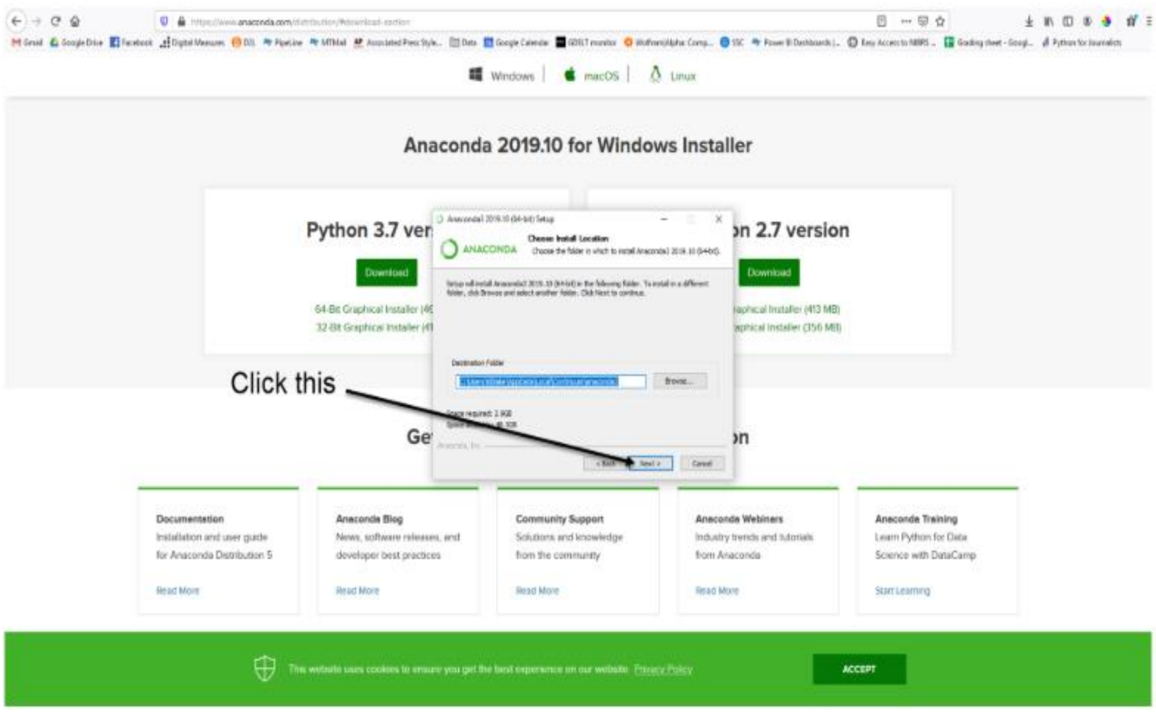
3. Now, you’ll see the license agreement. Click on ‘I Agree’.



4. You can install it for all users or just for yourself. If you want to install it for all users, you need administrator privileges.


Anaconda 2019.10 for Windows Installer
Select Installation Type
Just Me (recommended)
All Users (requires admin privileges)
Click this
Next >

5. Choose where you want to install it. Here, you can see the available space and how much you need.



Anaconda3 2019.10 (64-bit) Setup

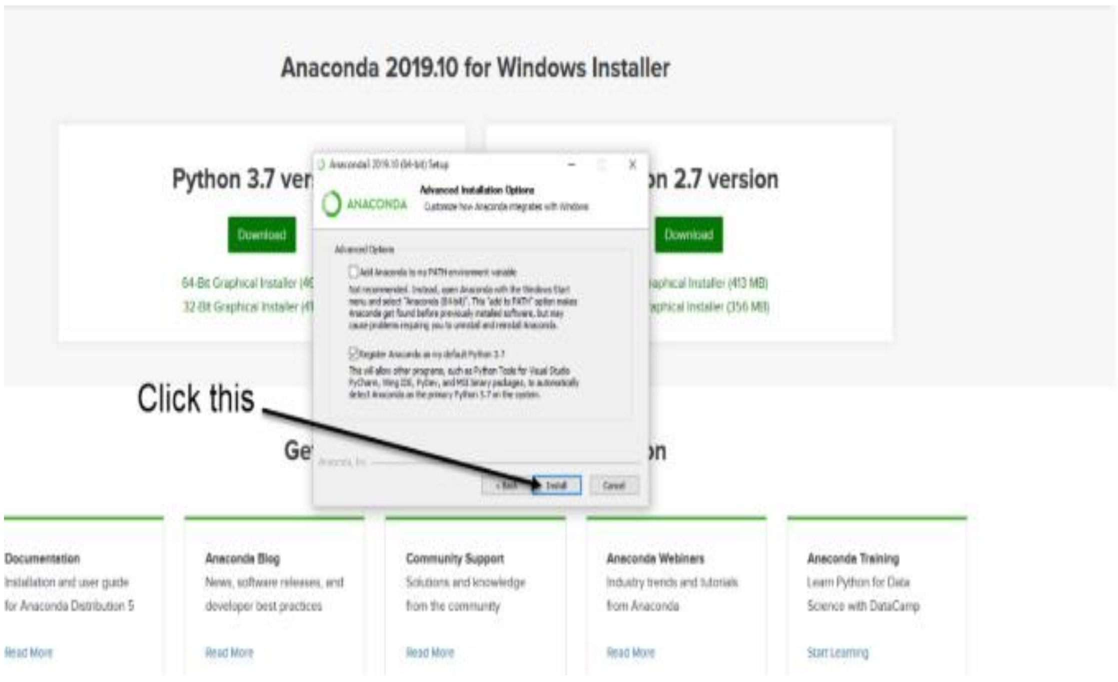
Choose Install Location
Choose the folder in which to install Anaconda3 2019.10 (64-bit).

Setup will install Anaconda3 2019.10 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

Space required: 2.9GB

6. Now, you'll get some advanced options. You can add Anaconda to your system's PATH environment variable, and register it as the primary system Python 3.7. If you add it to PATH, it will be found before any other installation. Click on 'Install'.



Anaconda3 2019.10 (64-bit) Setup

Advanced Installation Options
Customize how Anaconda integrates with Windows

Advanced Options

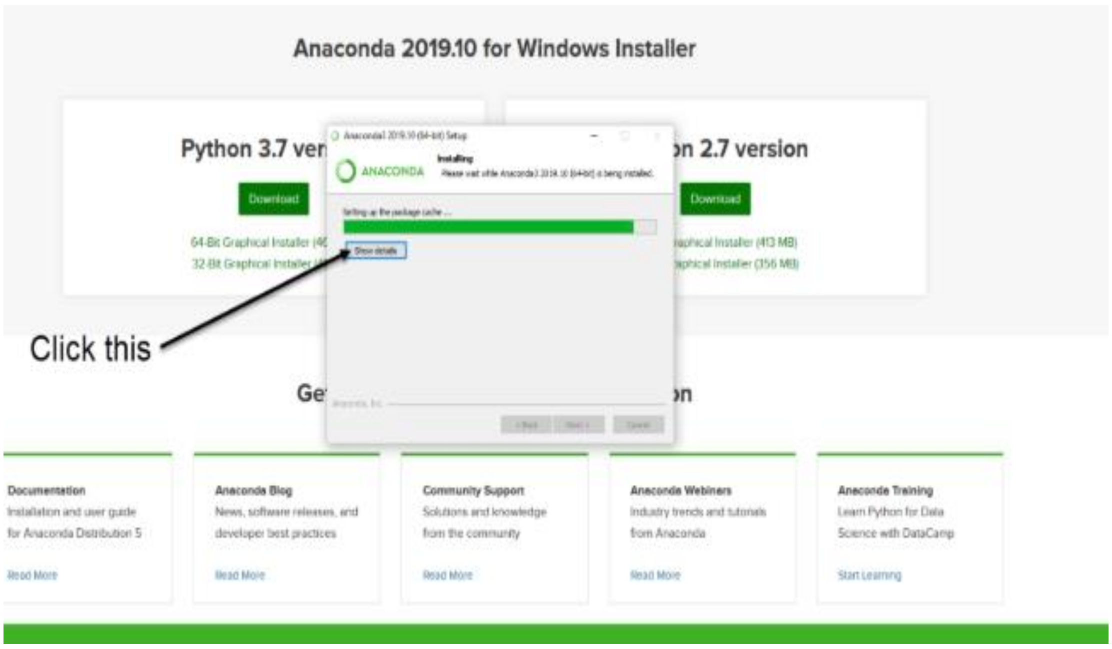
Add Anaconda to my PATH environment variable

Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

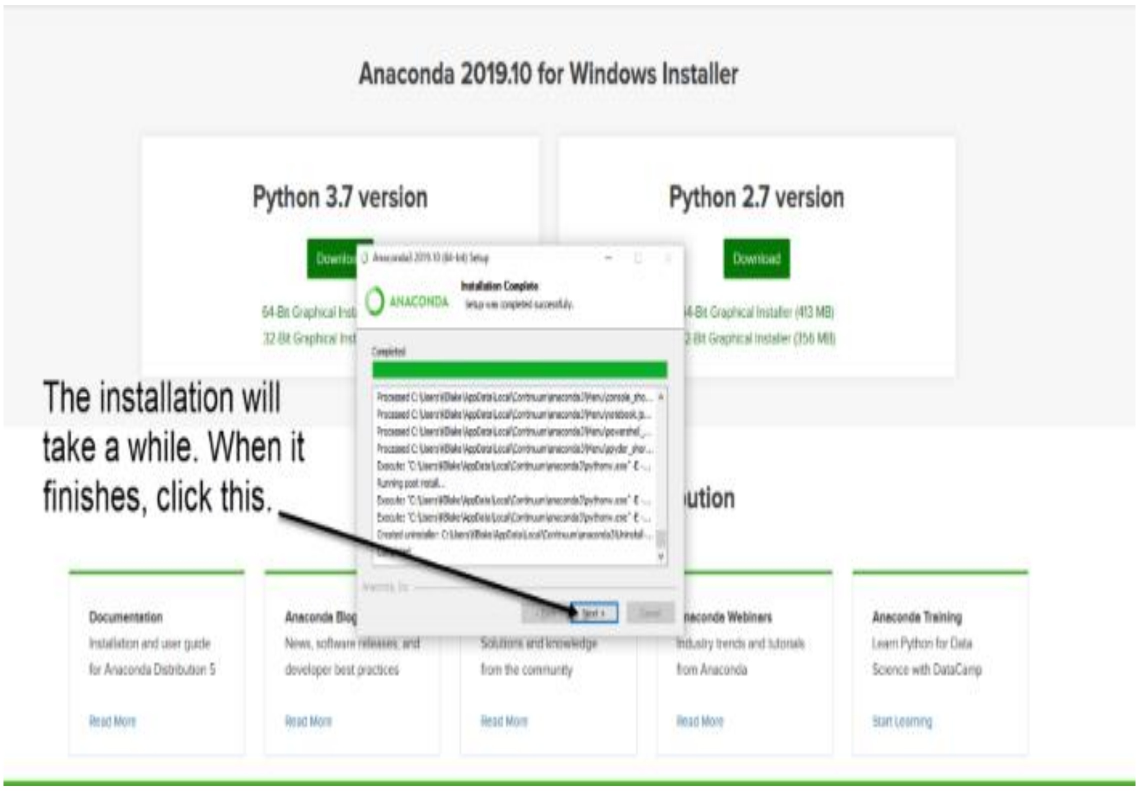
Register Anaconda as my default Python 3.7

This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.7 on the system.

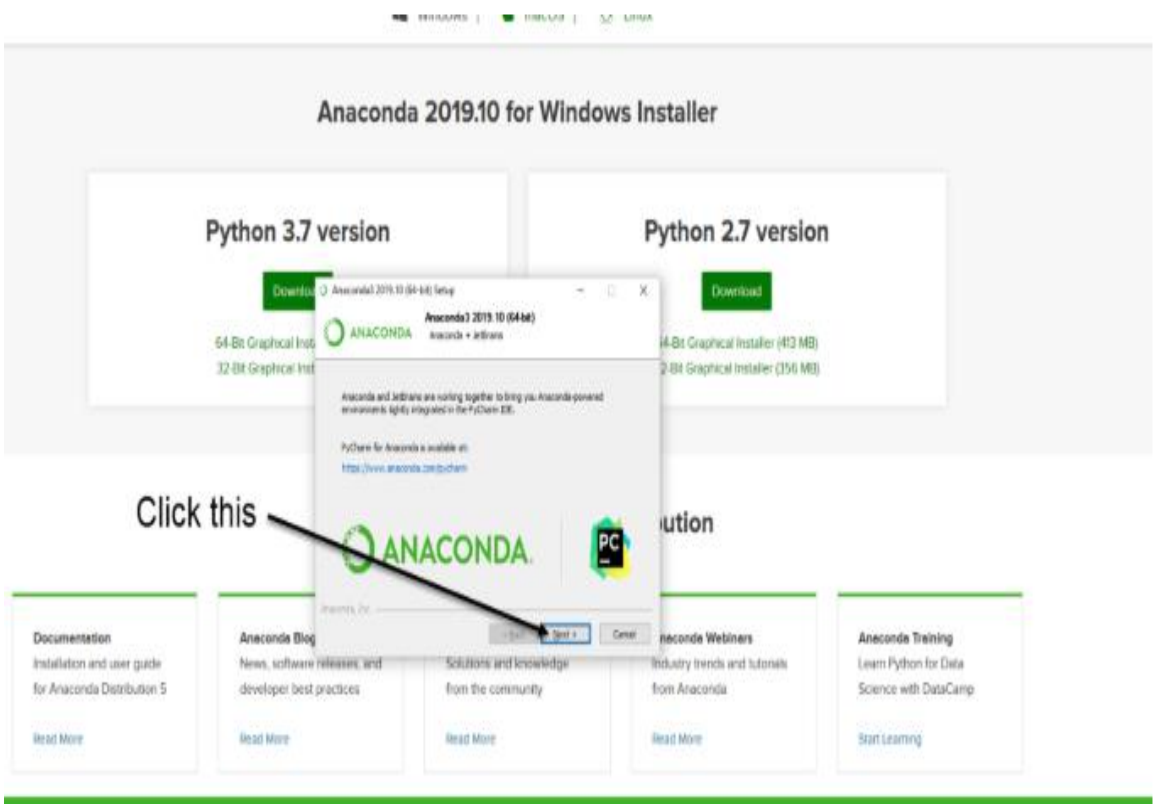
7. It will unpack some packages and extract some files on your machine. This will take a few minutes.



8. The installation is complete. Click Next.



9. This screen will inform you about PyCharm. Click Next.


Anaconda3 2019.10 (64-bit) Setup

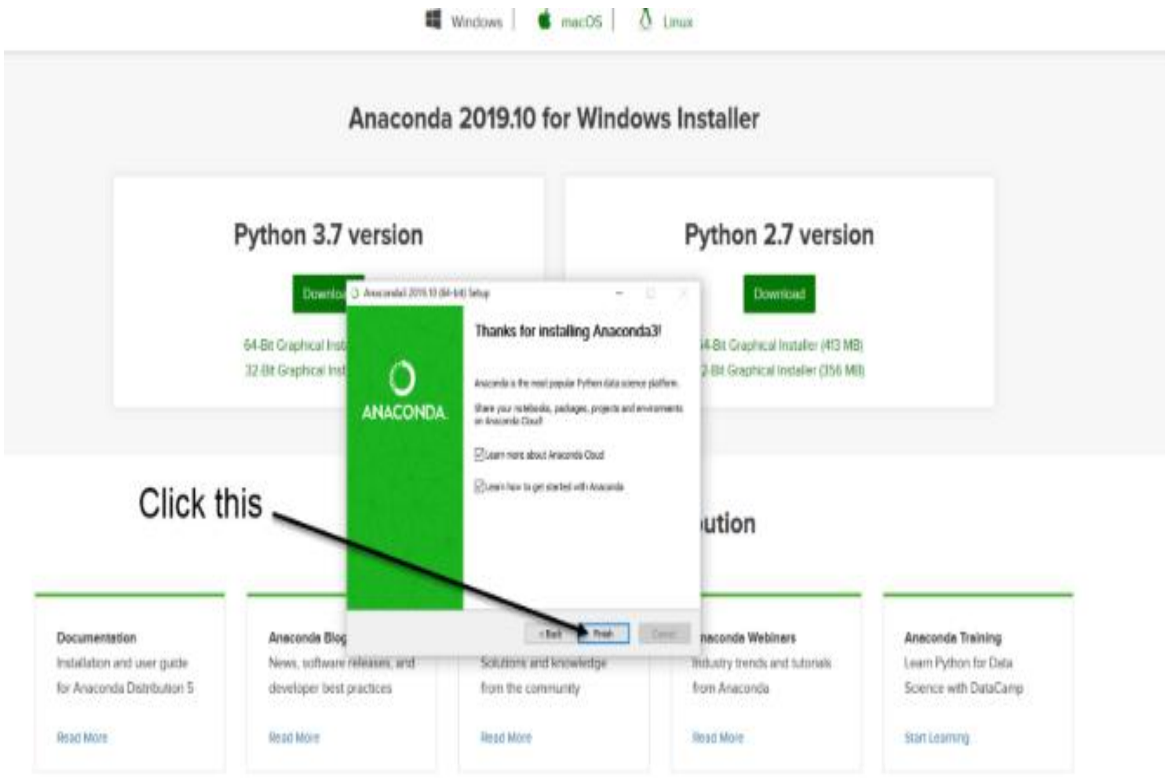
Anaconda3 2019.10 (64-bit)
Anaconda + JetBrains

Anaconda and JetBrains are working together to bring you Anaconda-powered environments tightly integrated in the PyCharm IDE.

PyCharm for Anaconda is available at:
https://www.anaconda.com/pycharm

Windows | macOS | Linux
Anaconda 2019.10 for Windows Installer
Python 3.7 version
Python 2.7 version
Download
64-Bit Graphical Installer (413 MB)
32-Bit Graphical Installer (356 MB)
Click this
ANACONDA
< Back
Next >
Cancel
Documentation
Installation and user guide for Anaconda Distribution 5
Read More
Anaconda Blog
News, software releases, and developer best practices
Solutions and knowledge from the community
Anaconda Webinars
Industry trends and tutorials from Anaconda
Anaconda Training
Learn Python for Data Science with DataCamp
Start Learning

10. The installation is complete. You can choose to get more information about Anaconda cloud and how to get started with Anaconda. Click Finish.


Anaconda3 2019.10 (64-bit) Setup

Thanks for installing Anaconda3!

Anaconda is the most popular Python data science platform.

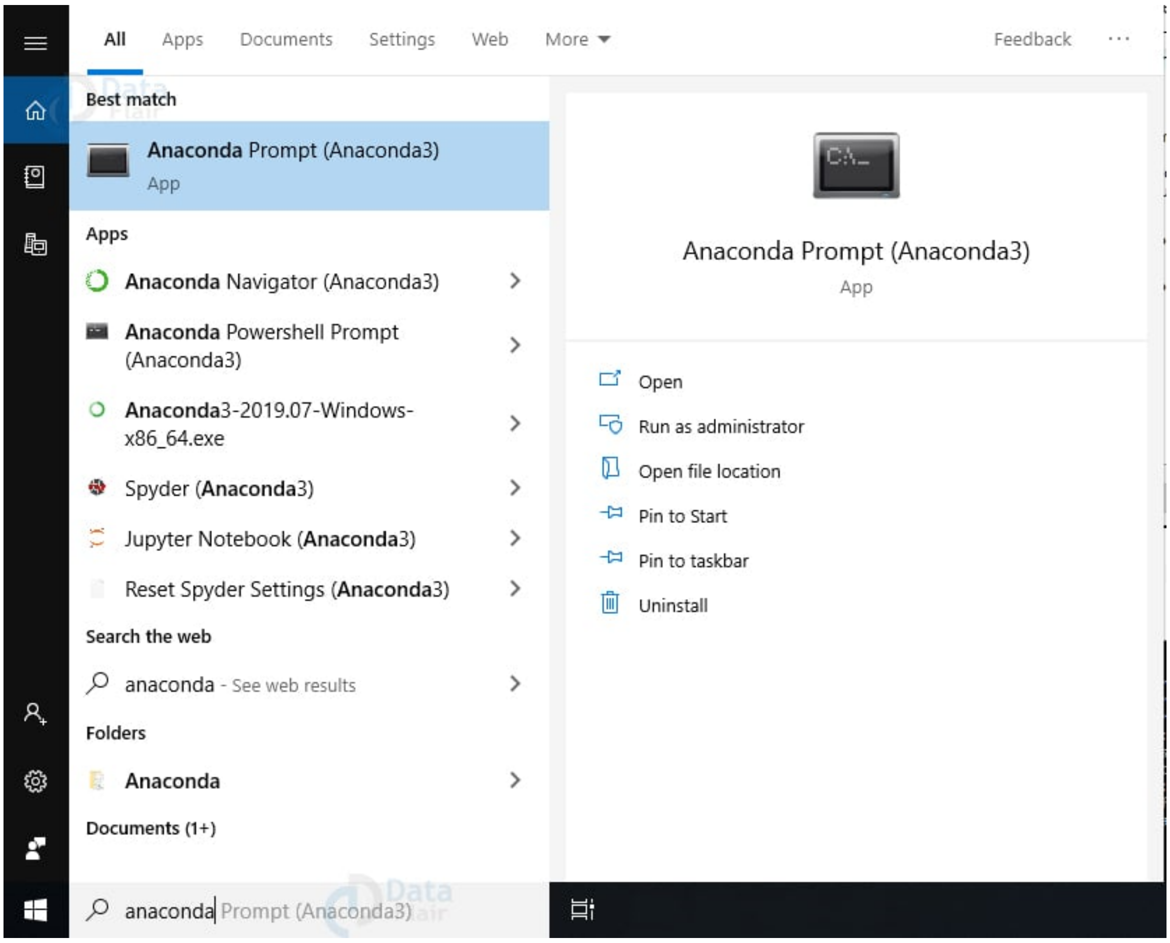
Share your notebooks, packages, projects and environments on Anaconda Cloud!

☑ Learn more about Anaconda Cloud

☑ Learn how to get started with Anaconda

Windows | macOS | Linux
Anaconda 2019.10 for Windows Installer
Python 3.7 version
Python 2.7 version
Download
64-Bit Graphical Installer (413 MB)
32-Bit Graphical Installer (356 MB)
Click this
ANACONDA
< Back
Finish
Cancel
Documentation
Installation and user guide for Anaconda Distribution 5
Read More
Anaconda Blog
News, software releases, and developer best practices
Solutions and knowledge from the community
Anaconda Webinars
Industry trends and tutorials from Anaconda
Anaconda Training
Learn Python for Data Science with DataCamp
Start Learning

11. If you search for Anaconda now, you will see the following options:


All
Apps
Documents
Settings
Web
More
Feedback
Best match
Anaconda Prompt (Anaconda3)
App
Apps
Anaconda Navigator (Anaconda3)
Anaconda Powershell Prompt (Anaconda3)
Anaconda3-2019.07-Windows-x86_64.exe
Spyder (Anaconda3)
Jupyter Notebook (Anaconda3)
Reset Spyder Settings (Anaconda3)
Search the web
anaconda - See web results
Folders
Anaconda
Documents (1+)
anaconda Prompt (Anaconda3)
Anaconda Prompt (Anaconda3)
App
Open
Run as administrator
Open file location
Pin to Start
Pin to taskbar
Uninstall

PYTHON LANGUAGE:

Python is an interpreter, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding; make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python is a dynamic, high-level, free open source, and interpreted programming language. It supports object-oriented programming as well as procedural-oriented programming. In Python, we don’t need to declare the type of variable because it is a dynamically typed language. For example, x = 10 Here, x can be anything such as String, int, etc.

Features in Python:

There are many features in Python, some of which are discussed below as follows:

1. Free and Open Source

Python language is freely available at the official website and you can download it from the given download link below click on the **Download Python** keyword. Download Python Since it is open-source, this means that source code is also available to the public. So, you can download it, use it as well as share it.

2. Easy to code

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, JavaScript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

3. Easy to Read

As you will see, learning Python is quite simple. As was already established, Python’s syntax is really straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

4. Object-Oriented Language

One of the key features of Python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, object encapsulation, etc.

5. GUI Programming Support

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.

6. High-Level Language

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

7. Extensible feature

Python is an **Extensible** language. We can write some Python code into C or C++ language and also we can compile that code in C/C++ language.

8. Easy to Debug

Excellent information for mistake tracing. You will be able to quickly identify and correct the majority of your program's issues once you understand how to interpret Python's error traces. Simply by glancing at the code, you can determine what it is designed to perform.

9. Python is a Portable language

Python language is also a portable language. For example, if we have Python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

10. Python is an integrated language

Python is also an integrated language because we can easily integrate Python with other languages like C, C++, etc.

11. Interpreted Language:

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile Python code this makes it easier to debug our code. The source code of Python is converted into an immediate form called **byte code**.

12. Large Standard Library

Python has a large standard library that provides a rich set of modules and functions so you do not have to write your own code for every single thing. There are many libraries present in Python such as regular expressions, unit-testing, web browsers, etc.

13. Dynamically Typed Language

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

14. Frontend and backend development

With a new project py script, you can run and write Python codes in HTML with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in Python like JavaScript. Backend is the strong forte of Python it's extensively used for this work cause of its frameworks like Django and Flask.

15. Allocating Memory Dynamically

In Python, the variable data type does not need to be specified. The memory is automatically allocated to a variable at runtime when it is given a value. Developers do not need to write int y = 18 if the integer value 15 is set to y. You may just type y=18.

LIBRARIES/PACKGES: -

Tensor flow

Tensor Flow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

TensorFlow offers a high-level API called Keras, which simplifies the process of building and training neural networks. With Keras, users can quickly prototype and experiment with different architectures, loss functions, and optimization algorithms, while still retaining the flexibility to customize their models as needed. Keras also provides pre-trained models and utilities for common tasks such as image classification, text generation, and sequence-to-sequence translation, enabling rapid development of AI applications.

Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Pandas is a versatile and powerful library for data manipulation and analysis in Python. Whether you're cleaning, transforming, exploring, or summarizing data, Pandas provides the tools and functionalities you need to tackle a wide range of data-related tasks efficiently and effectively. Its intuitive interface, extensive documentation, and active community make it an indispensable tool for data scientists, analysts, and researchers across various domains.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object-oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provide a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

Its extensive suite of functionalities covers classification, regression, clustering, dimensionality reduction, and model selection, all implemented with a consistent and intuitive API. Beyond its technical prowess, scikit-learn boasts a robust community and comprehensive documentation, making it accessible to both novices and seasoned practitioners. With capabilities spanning data preprocessing, feature extraction, model evaluation, and seamless integration with other Python libraries, it simplifies the end-to-end process of developing machine learning solutions. Its scalability, open-source nature, and permissive licensing further underscore its significance, rendering it indispensable for researchers, academics, and industry professionals alike.

8. SYSTEM TESTING

System testing, also referred to as system-level tests or system-integration testing, is the process in which a quality assurance (QA) team evaluates how the various components of an application interact together in the full, integrated system or application. System testing verifies that an application performs tasks as designed. This step, a kind of black box testing, focuses on the functionality of an application. System testing, for example, might check that every kind of user input produces the intended output across the application.

System testing is a pivotal phase in the software development lifecycle, encompassing a series of rigorous evaluations aimed at validating the overall functionality, performance, and reliability of a software system. This testing phase serves to ensure that the system meets its specified requirements and is ready for deployment in a production environment. System testing involves activities such as test planning, test case design, test execution, defect reporting and tracking, regression testing, performance testing, and acceptance testing. Test planning involves creating a comprehensive test plan outlining objectives, scope, and timelines. Test case design involves creating test scenarios and writing test cases covering positive and negative scenarios. During test execution, test cases are executed, and defects are logged into a defect tracking system for resolution. Regression testing ensures that changes to the system do not introduce new defects, while performance testing assesses the system's responsiveness and scalability. Acceptance testing validates the system against user acceptance criteria. Overall, system testing plays a critical role in ensuring the quality, reliability, and performance of software systems, mitigating risks, and enhancing user satisfaction.

Phases of system testing:

System testing comprises several sequential phases crucial for ensuring the quality and reliability of software systems. The initial phase involves meticulous test planning, where objectives, scope, and resources are outlined to provide a roadmap for subsequent activities. Test design follows, where test cases and scenarios are meticulously crafted to cover various system functionalities. In the subsequent test execution phase, these test cases are executed, and results are recorded, with defects being logged and managed through a defect tracking system. Defects are then resolved through collaboration between testers and developers, with the system undergoing retesting to validate fixes. Test reporting summarises findings and communicates outcomes to stakeholders, aiding decision-making and providing insights into system quality. Finally, test closure ensures that all objectives are met, with lessons learned documented for future improvements. Through these systematic phases, system testing ensures the reliability, functionality, and performance of software systems, ultimately enhancing customer satisfaction and business success.

A video tutorial about this test level. System testing examines every component of an application to make sure that they work as a complete and unified whole. A QA team typically conducts system testing after it checks individual modules with functional or user-story testing and then each component through integration testing.

If a software build achieves the desired results in system testing, it gets a final check via acceptance testing before it goes to production, where users consume the software. An app-dev team logs all defects, and establishes what kinds and amount-of defects are tolerable.

8.1 Software Testing Strategies:

Optimization of the approach to testing in software engineering is the best way to make it effective. A software testing strategy defines what, when, and how to do whatever is necessary to make an end-product of high quality.

Software testing strategies encompass a range of approaches and techniques aimed at ensuring the quality, reliability, and effectiveness of software products. These strategies are essential components of the software development lifecycle, helping to identify defects, mitigate risks, and validate that the software meets user requirements.

Usually, the following software testing strategies and their combinations are used to achieve this major objective:

Static Testing:

The early-stage testing strategy is static testing: it is performed without actually running the developing product. Basically, such desk-checking is required to detect bugs and issues that are present in the code itself. Such a check-up is important at the pre-deployment stage as it helps avoid problems caused by errors in the code and software structure deficits.

Static testing strategies involve evaluating software artifacts without executing the code. These strategies focus on identifying defects, inconsistencies, and opportunities for improvement in the early stages of the software development lifecycle. By analyzing the software's documentation, source code, and other static artifacts, static testing techniques help improve quality, reliability, and maintainability while reducing the cost and effort associated with defect resolution.

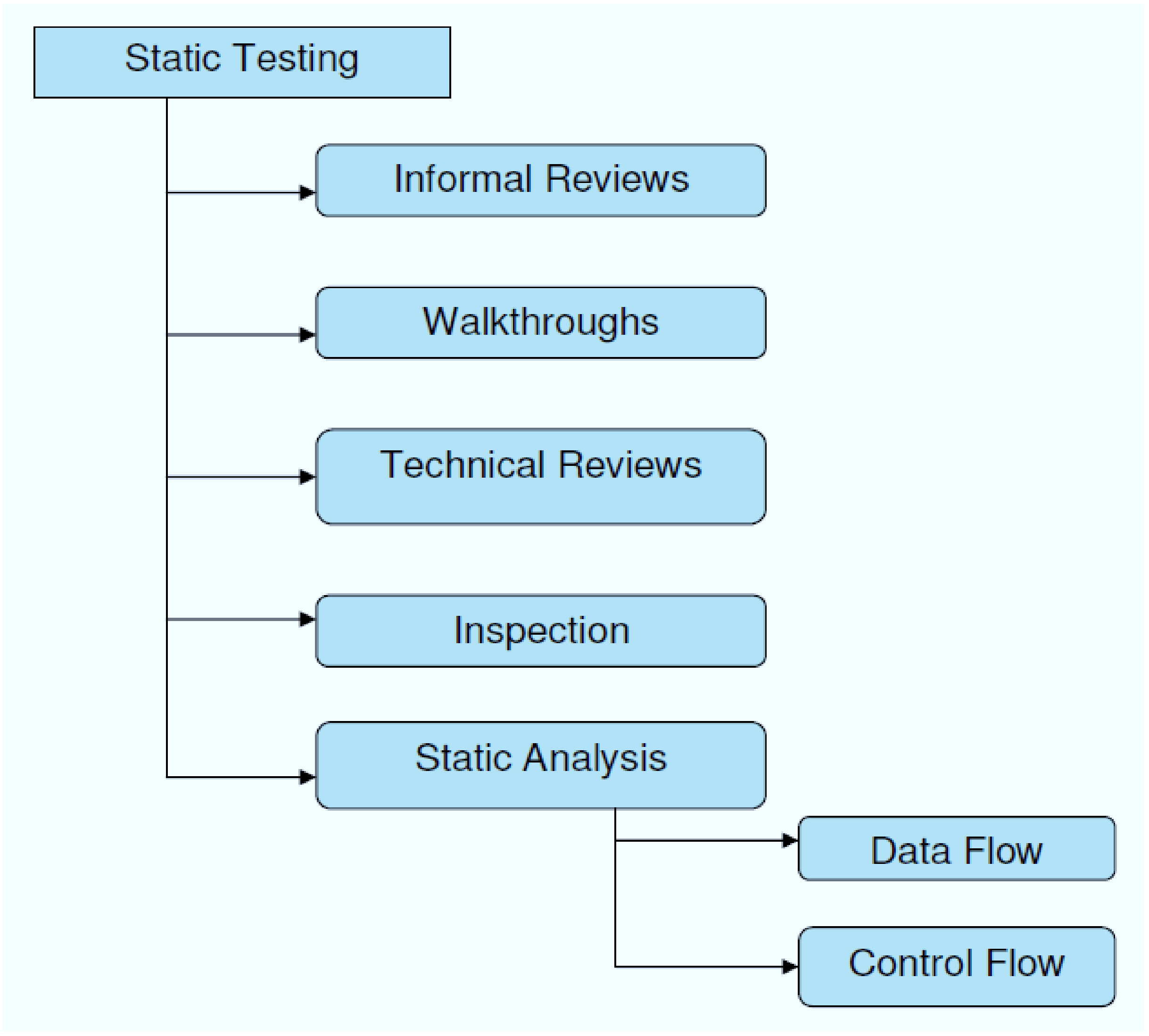


Figure 13: Static Testing flow

Static testing is a crucial aspect of software development that involves examining software artifacts without executing them. Unlike dynamic testing, which involves running the software, static testing focuses on reviewing documentation, source code, design specifications, and other development artifacts to identify defects early in the development process. This proactive approach helps in improving the quality of the software, reducing costs, and ensuring that the final product meets the desired requirements.

One common form of static testing is code review or peer review. In this process, developers inspect each other's code to identify issues such as coding errors, deviations from coding standards, potential security vulnerabilities, and design flaws. Code reviews can be formal, with specific checklists and guidelines, or informal, where developers discuss code changes and improvements.

Another static testing technique is walkthroughs and inspections, where a group of stakeholders examines software artifacts such as requirements documents, design specifications, and test plans to identify problems and discrepancies. These sessions can involve developers, testers, business analysts, and other relevant parties to ensure that all aspects of the software are thoroughly reviewed.

Structural Testing:

It is not possible to effectively test software without running it. Structural testing, also known as white-box testing, is required to detect and fix bugs and errors emerging during the pre-production stage of the software development process. At this stage, unit testing based on the software structure is performed using regression testing. In most cases, it is an automated process working within the test automation framework to speed up the development process at this stage. Developers and QA engineers have full access to the software’s structure and data flows (data flows testing), so they could track any changes (mutation testing) in the system’s behavior by comparing the tests’ outcomes with the results of previous iterations (control flow testing).

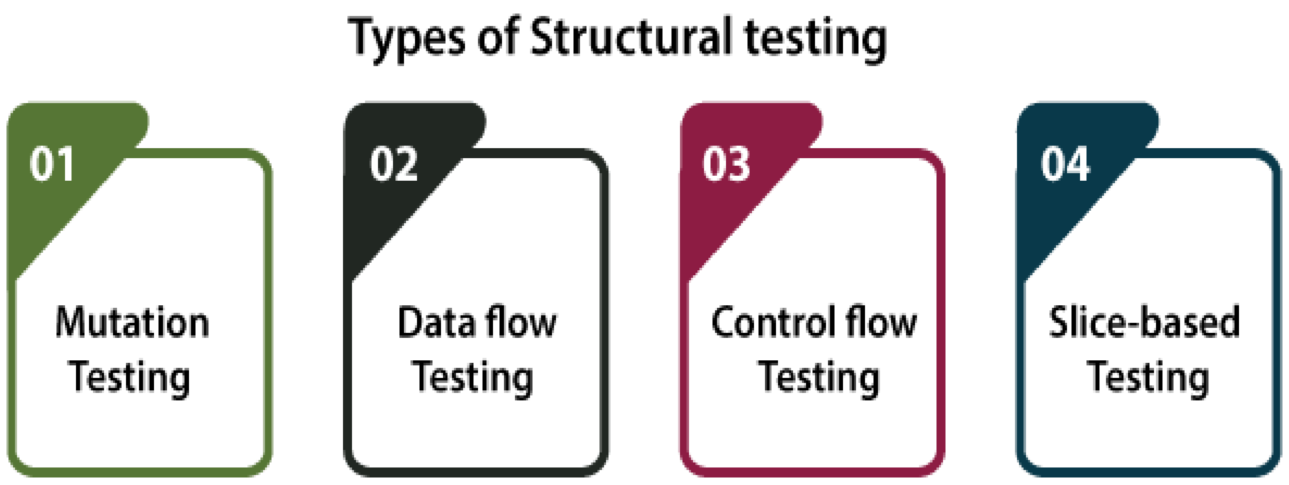


Figure 14: Types of Structural Testing

Mutation Testing: Mutation testing is a technique used to assess the effectiveness of test cases by introducing small changes, or mutations, into the codebase. Each mutation represents a potential fault that could occur in the software. The test suite is then executed against these mutated versions of the code, aiming to detect whether the tests can identify and distinguish between the original program and the mutated versions. If a test case fails to detect a mutation, it indicates a potential weakness in the test suite's ability to detect similar faults in the actual codebase, highlighting areas for improvement in test coverage and effectiveness.

Data flow Testing: Data flow testing is a white-box testing technique that focuses on identifying and analyzing the flow of data within a software application. It aims to uncover potential vulnerabilities and defects related to how data is processed, transformed, and stored throughout the system. By tracing the paths of data from its source to its destination, data flow testing helps ensure that information is handled correctly and securely. This technique involves creating test cases based on data dependencies, such as variable definitions, assignments, and references, to validate the proper flow of data and detect anomalies or inconsistencies in data handling logic.

Control flow Testing: Control flow testing is a white-box testing technique that evaluates the logical flow of control within a software program. It aims to identify and test the various control structures, such as loops, conditionals, and branches, to ensure that all possible control paths are exercised during testing. Test cases are designed to cover different control flow scenarios, including true and false conditions, loop iterations, and exception handling. By systematically testing control flow paths, this technique helps uncover errors, dead code, and logic flaws that may affect the program's behavior and reliability, thereby enhancing test coverage and software quality.

Slice-based Testing: Slice-based testing is a white-box testing technique that focuses on reducing the complexity of test cases by selecting a subset of the program's code, known as a "slice," for testing. This subset contains statements and variables relevant to the specific functionality being tested, as well as any dependencies required for its execution. By isolating the relevant code segments, slice-based testing aims to simplify test case creation and execution, making it easier to identify defects and verify the correctness of the software. This technique helps improve test coverage and efficiency by targeting the most critical parts of the codebase, thereby enhancing overall testing effectiveness and reducing the effort required for test maintenance.

Statement Coverage: This technique ensures that every statement in the code is executed at least once during testing. Test cases are designed to exercise different parts of the code, including loops, conditional statements, and error-handling routines, to verify their correctness.

Branch Coverage: Branch coverage aims to test all possible outcomes of conditional statements, including both true and false branches. Test cases are created to traverse each branch of the code, ensuring that all decision points are evaluated under different conditions.

Path Coverage: Path coverage involves testing every possible path through the software's control flow graph, from the entry point to the exit point. This technique ensures that all potential execution paths, including loops and nested conditionals, are exercised during testing.

Condition Coverage: Condition coverage focuses on testing the Boolean expressions within conditional statements to ensure that they evaluate to both true and false values. Test cases are designed to evaluate each condition independently, as well as in combination with other conditions, to validate their behavior.

Loop Coverage: Loop coverage aims to test the different iterations of loops within the code, including zero iterations, single iterations, and multiple iterations. Test cases are created to exercise various loop scenarios, such as empty loops, loops with a single iteration, and loops with multiple iterations, to verify their correctness.

Data Flow Coverage: Data flow coverage analyzes the flow of data within the software to ensure that variables are initialized, used, and updated correctly throughout the program. Test cases are designed to trace the propagation of data through different parts of the code, identifying potential data dependencies and inconsistencies.

Structural testing is essential for ensuring the reliability, maintainability, and security of software systems. By thoroughly examining the internal structure of the code and verifying its correctness through various testing techniques, developers can identify and fix defects early in the development process, reducing the likelihood of bugs occurring in production. Additionally, structural testing helps improve code quality, readability, and scalability, making it easier to maintain and enhance software systems over time. Overall, structural testing plays a crucial role in the software development lifecycle by ensuring that software products meet the desired quality standards and requirements.

Behavioral Testing:

The final stage of testing focuses on the software's reactions to various activities rather than on the mechanisms behind these reactions. In other words, behavioral testing, also known as black-box testing, presupposes running numerous tests, mostly manual, to see the product from the user's point of view. QA engineers usually have some specific information about a business or other purposes of the software ('the black box') to run usability tests, for example, and react to bugs as regular users of the product will do. Behavioral testing also may include automation (regression tests) to eliminate human error if repetitive activities are required. For example, you may need to fill 100 registration forms on the website to see how the product copes with such an activity, so the automation of this test is preferable.

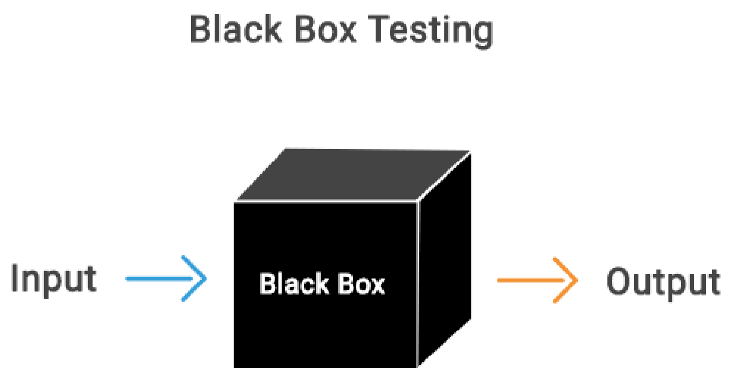


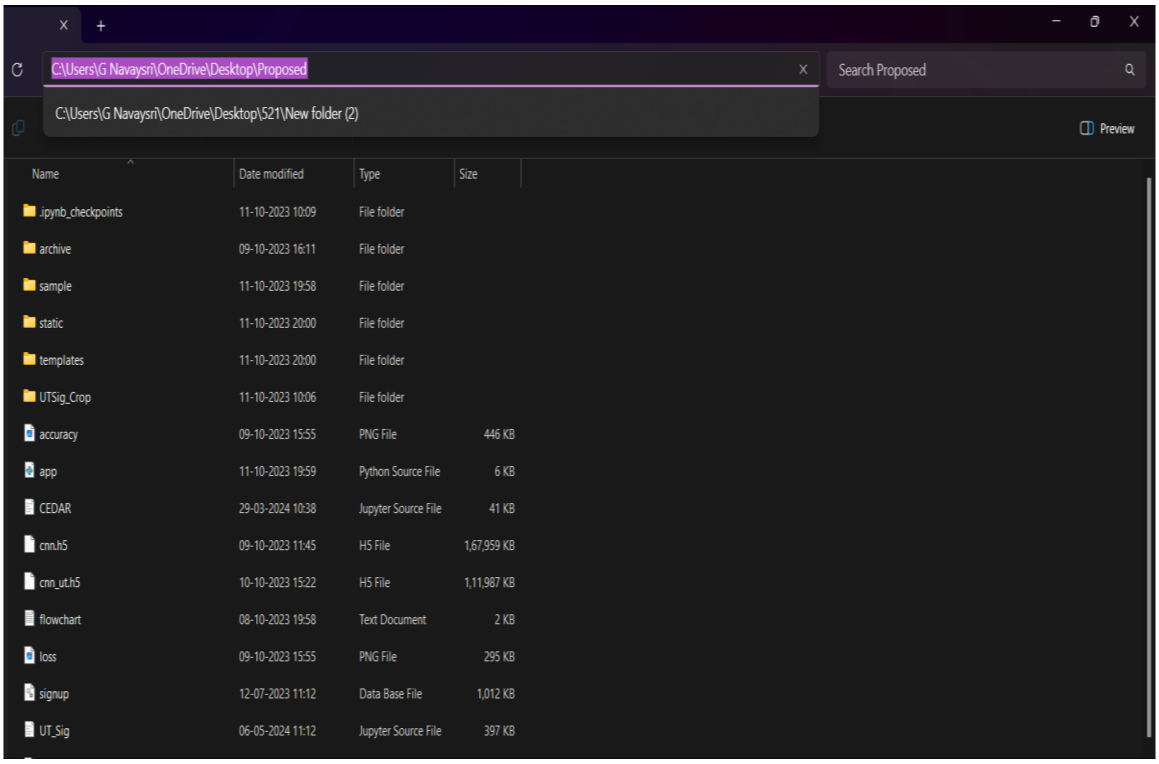
Figure 15: Block Box Testing

Behavioral testing is crucial for ensuring that the software behaves as expected under various conditions. It involves simulating user interactions and analyzing system responses to validate compliance with specifications. Functional testing verifies individual functions or features, while UI testing evaluates the graphical interface's usability and design. UX testing assesses the overall user experience, considering factors like satisfaction and efficiency. Usability testing specifically examines how easily users can accomplish tasks with the software. By employing these techniques, behavioral testing enhances software quality, reliability, and user satisfaction.

8.2 TEST CASES:

S.NO	INPUT	If available	If not available
1	User signup	User get registered into the application	There is no process
2	User sign in	User get login into the application	There is no process
3	Enter input for prediction	Prediction result displayed	There is no process

9. RESULT

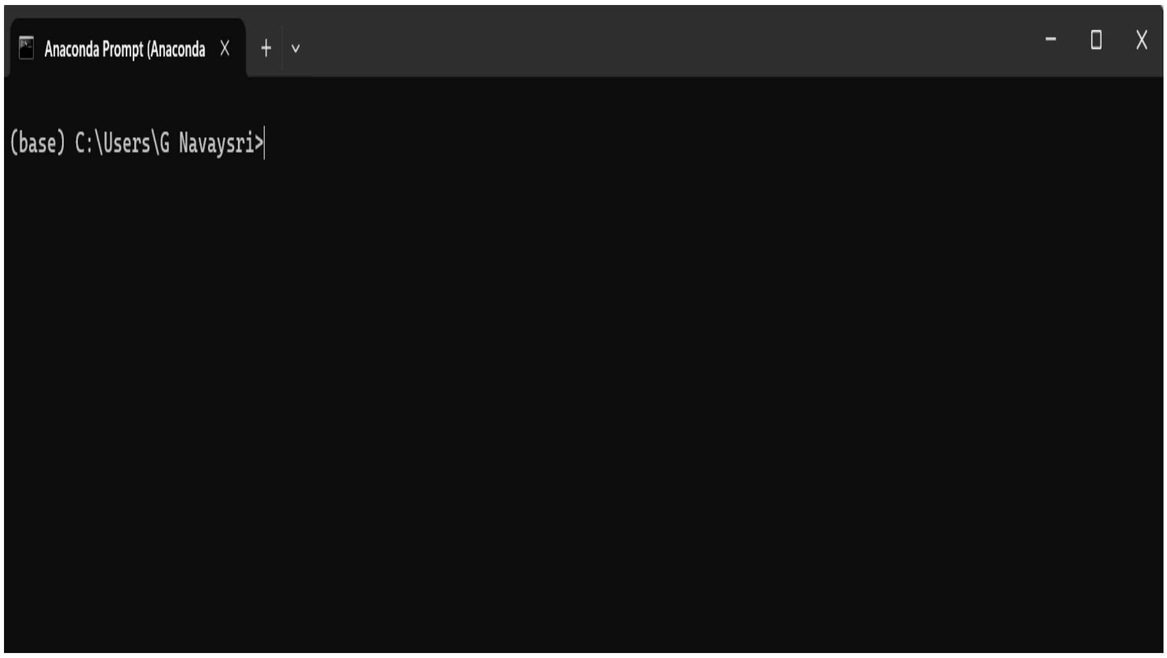


C:\Users\G Navaysri\OneDrive\Desktop\Proposed

C:\Users\G Navaysri\OneDrive\Desktop\521\New folder (2)

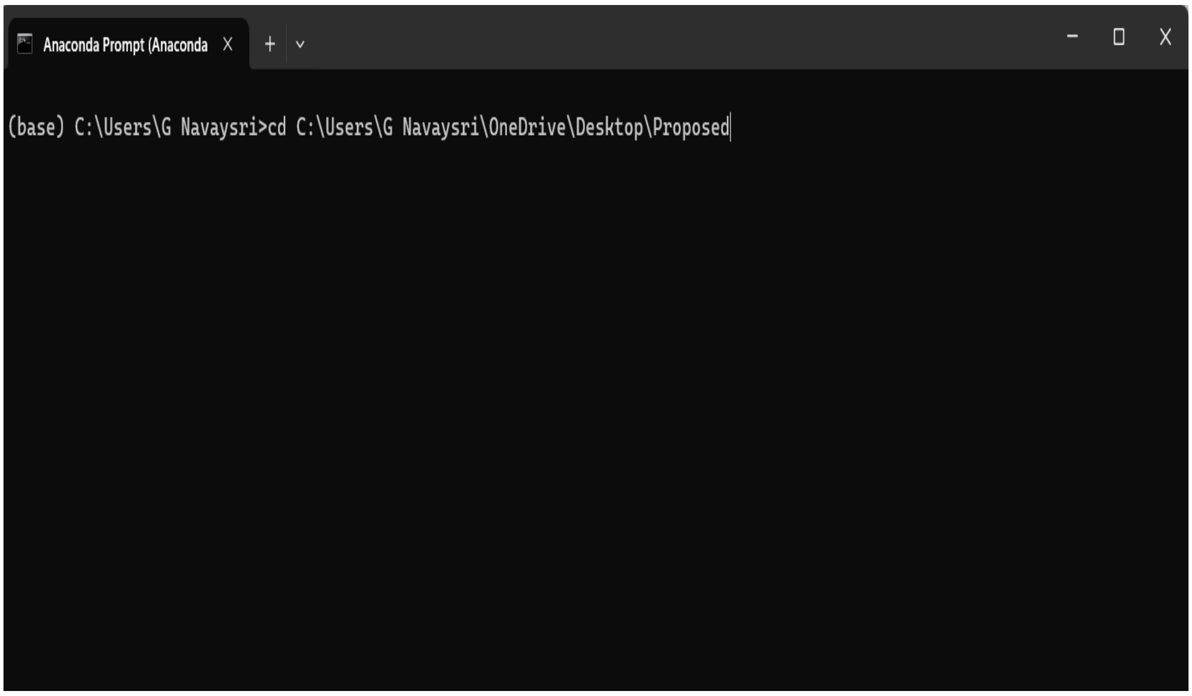
Name	Date modified	Type	Size
.ipynb_checkpoints	11-10-2023 10:09	File folder	
archive	09-10-2023 16:11	File folder	
sample	11-10-2023 19:58	File folder	
static	11-10-2023 20:00	File folder	
templates	11-10-2023 20:00	File folder	
UTSig_Crop	11-10-2023 10:06	File folder	
accuracy	09-10-2023 15:55	PNG File	446 KB
app	11-10-2023 19:59	Python Source File	6 KB
CEDAR	29-03-2024 10:38	Jupyter Source File	41 KB
cnn.h5	09-10-2023 11:45	H5 File	1,67,959 KB
cnn_ut.h5	10-10-2023 15:22	H5 File	1,11,987 KB
flowchart	08-10-2023 19:58	Text Document	2 KB
loss	09-10-2023 15:55	PNG File	295 KB
signup	12-07-2023 11:12	Data Base File	1,012 KB
UT_Sig	06-05-2024 11:12	Jupyter Source File	397 KB

Screen 1: Login's List



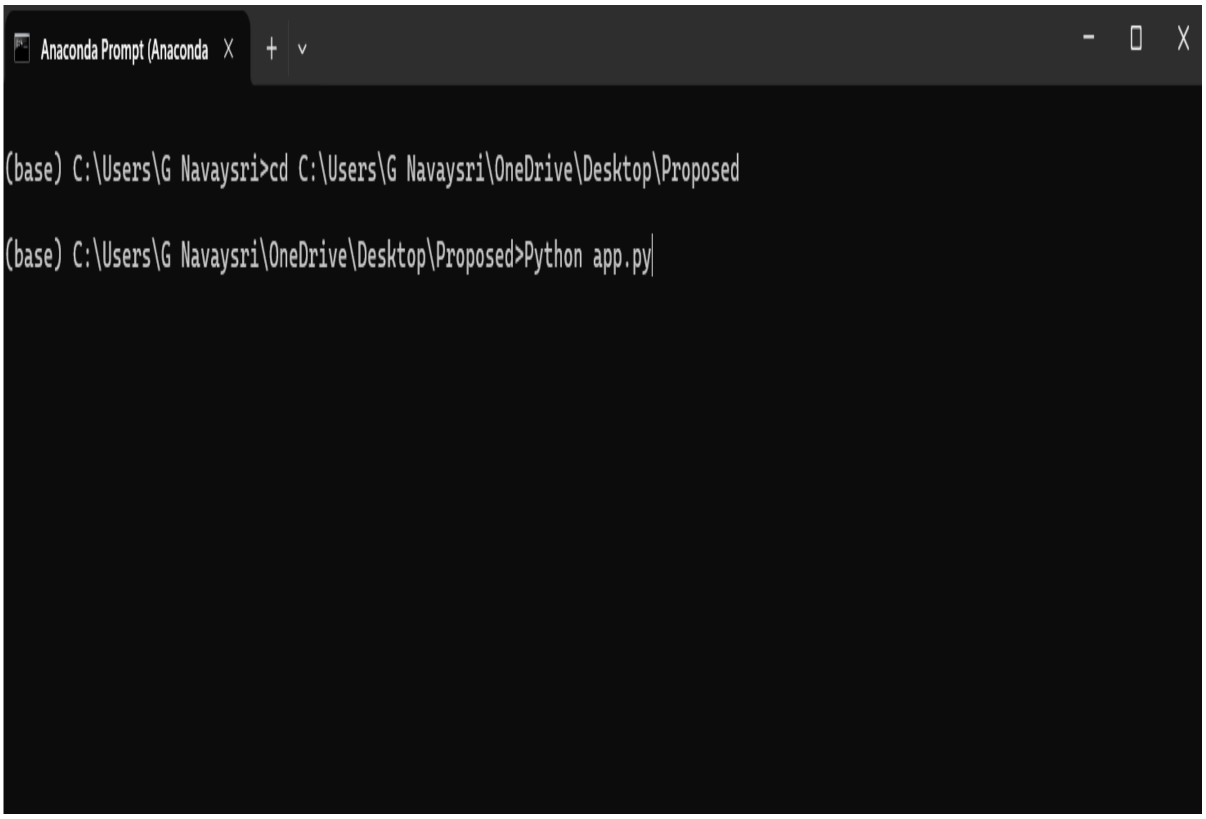
```
(base) C:\Users\G Navaysri>
```

Screen 2: Command Prompt



```
(base) C:\Users\G Navaysri>cd C:\Users\G Navaysri\OneDrive\Desktop\Proposed
```

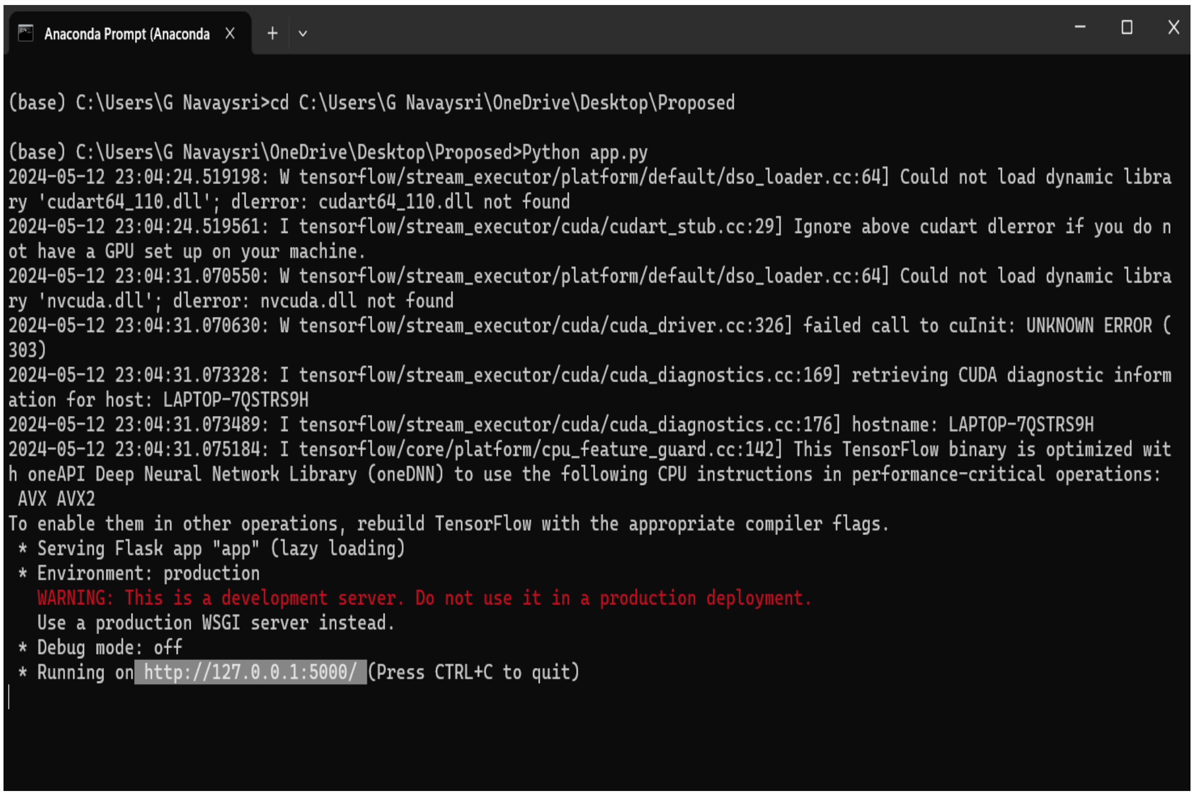
Screen 2.a: Command prompt with specified Address



```
(base) C:\Users\G Navaysri>cd C:\Users\G Navaysri\OneDrive\Desktop\Proposed

(base) C:\Users\G Navaysri\OneDrive\Desktop\Proposed>Python app.py
```

Screen 2.b: Application running on command prompt

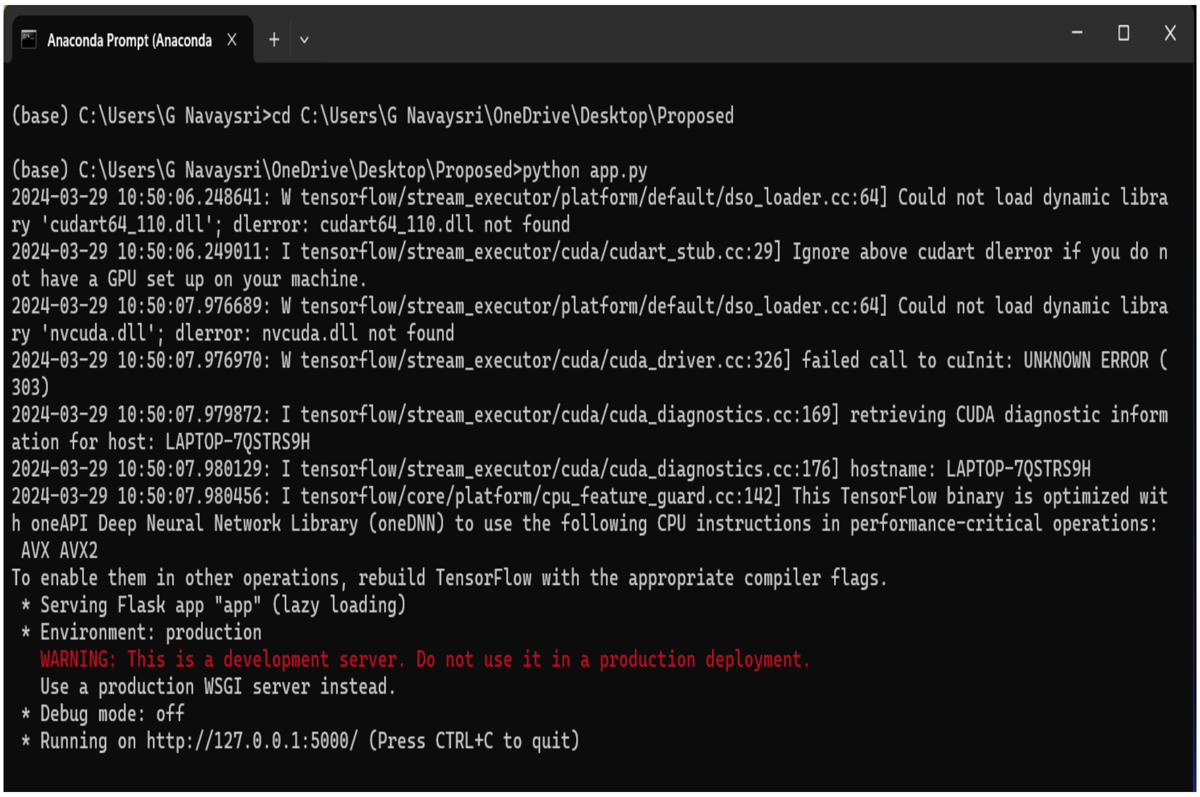


Anaconda Prompt (Anaconda

```
(base) C:\Users\G Navaysri>cd C:\Users\G Navaysri\OneDrive\Desktop\Proposed

(base) C:\Users\G Navaysri\OneDrive\Desktop\Proposed>Python app.py
2024-05-12 23:04:24.519198: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic libra
ry 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2024-05-12 23:04:24.519561: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do n
ot have a GPU set up on your machine.
2024-05-12 23:04:31.070550: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic libra
ry 'nvcuda.dll'; dlerror: nvcuda.dll not found
2024-05-12 23:04:31.070630: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOWN ERROR (
303)
2024-05-12 23:04:31.073328: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic inform
ation for host: LAPTOP-7QSTRS9H
2024-05-12 23:04:31.073489: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: LAPTOP-7QSTRS9H
2024-05-12 23:04:31.075184: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized wit
h oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:
 AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

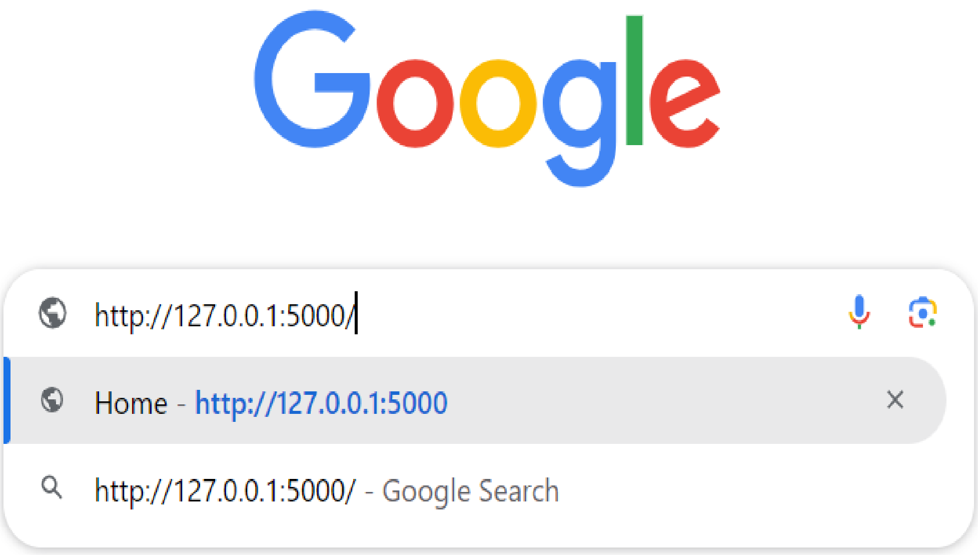
Screen 2.c: Getting Hyperlink Address



Anaconda Prompt (Anaconda

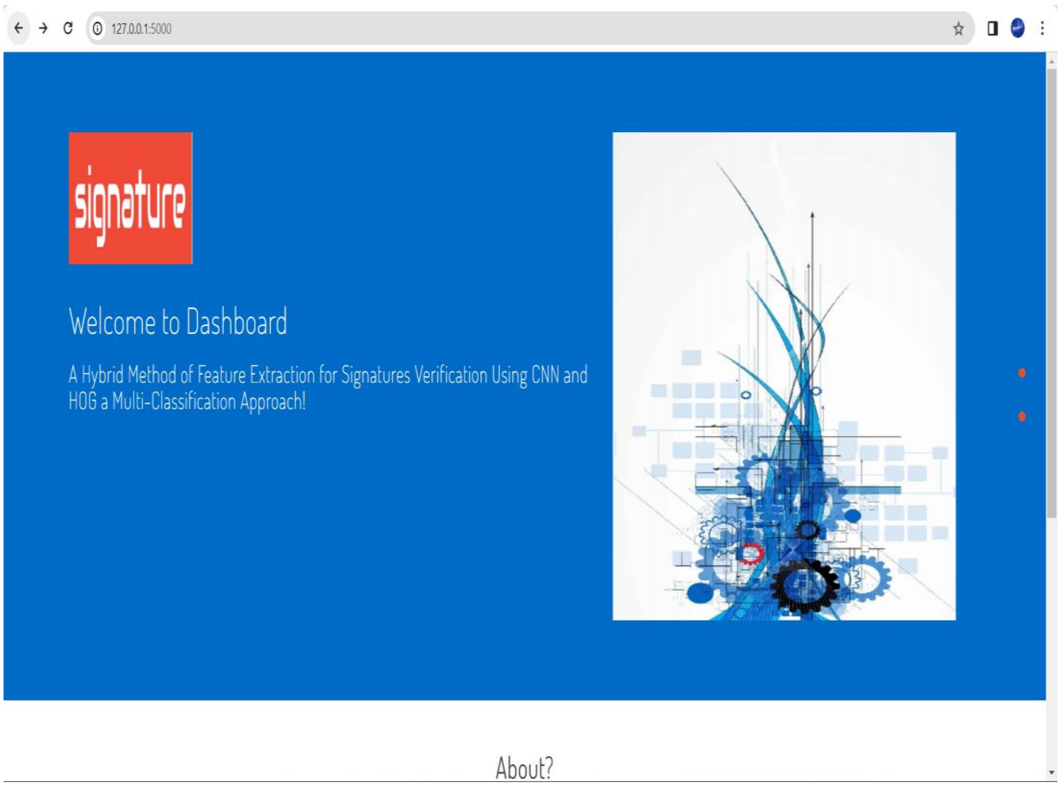
```
(base) C:\Users\G Navaysri>cd C:\Users\G Navaysri\OneDrive\Desktop\Proposed

(base) C:\Users\G Navaysri\OneDrive\Desktop\Proposed>python app.py
2024-03-29 10:50:06.248641: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic libra
ry 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2024-03-29 10:50:06.249011: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do n
ot have a GPU set up on your machine.
2024-03-29 10:50:07.976689: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic libra
ry 'nvcuda.dll'; dlerror: nvcuda.dll not found
2024-03-29 10:50:07.976970: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOWN ERROR (
303)
2024-03-29 10:50:07.979872: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic inform
ation for host: LAPTOP-7QSTRS9H
2024-03-29 10:50:07.980129: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: LAPTOP-7QSTRS9H
2024-03-29 10:50:07.980456: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized wit
h oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:
 AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```


Google
http://127.0.0.1:5000/
Home - http://127.0.0.1:5000
http://127.0.0.1:5000/ - Google Search

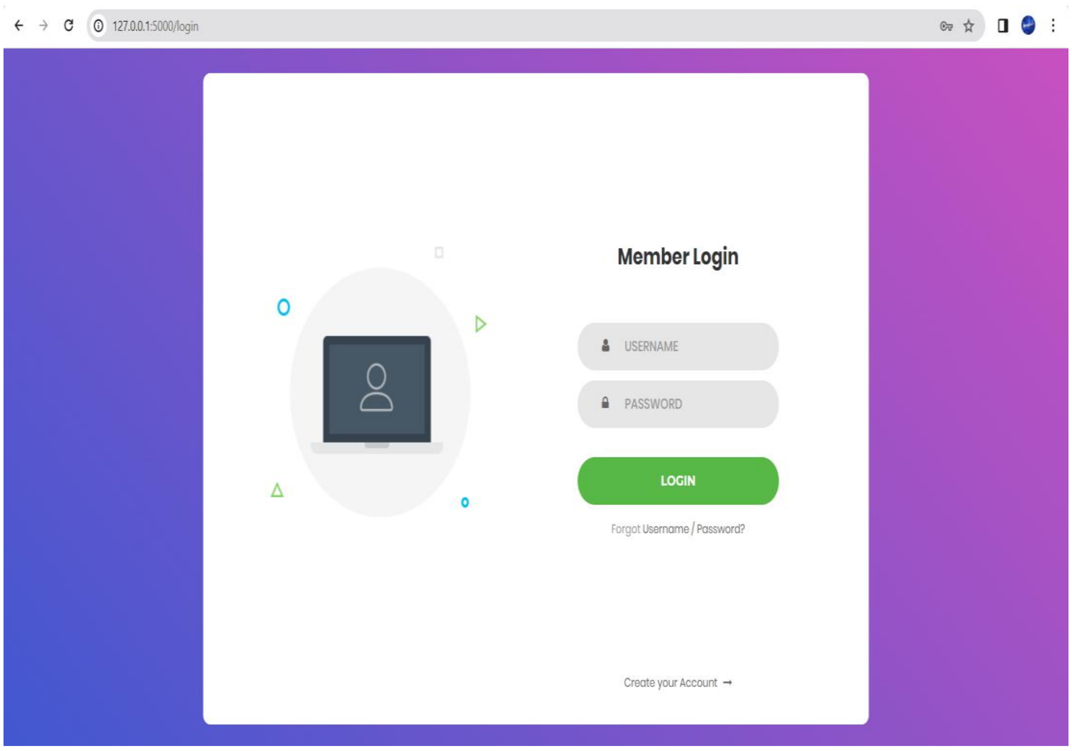
Screen 3: URL of server

1. WEBSITE


127.0.0.1:5000
signature
Welcome to Dashboard
A Hybrid Method of Feature Extraction for Signatures Verification Using CNN and HOG a Multi-Classification Approach!
About?

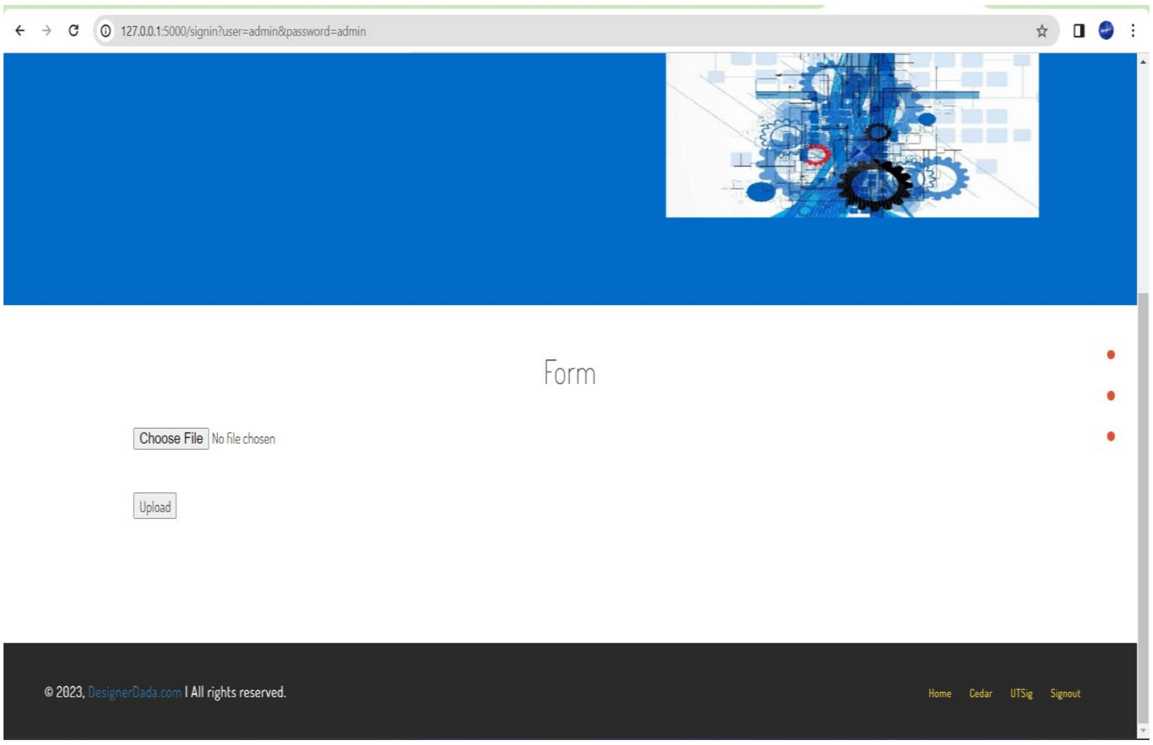
Screen 4: Home page of website

2. Login process



Screen 4.a: Register/Login process

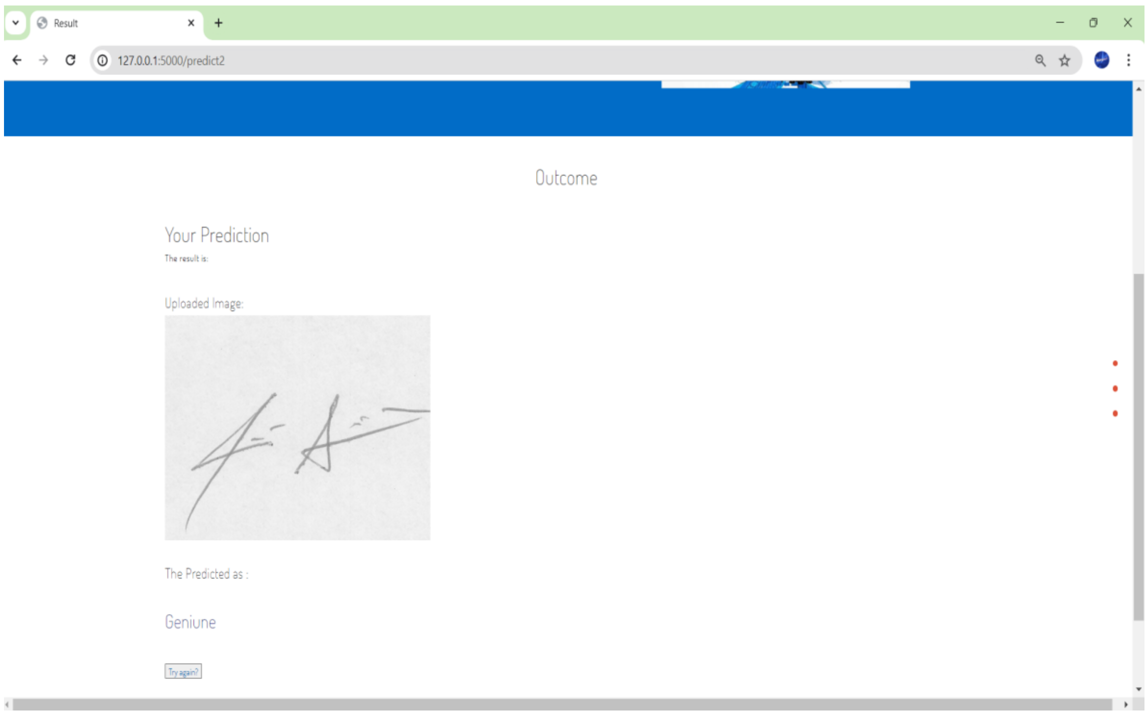
3. Upload Image


127.0.0.1:5000/signin?user=admin&password=admin
Form
Choose File No file chosen
Upload
© 2023, DesignerDada.com I All rights reserved.
Home Cedar UTSig Signout

Screen 4.b: Uploading Signature Image

4. Outcome

Genuine


Result

127.0.0.1:5000/predict2

Outcome

Your Prediction

The result is:

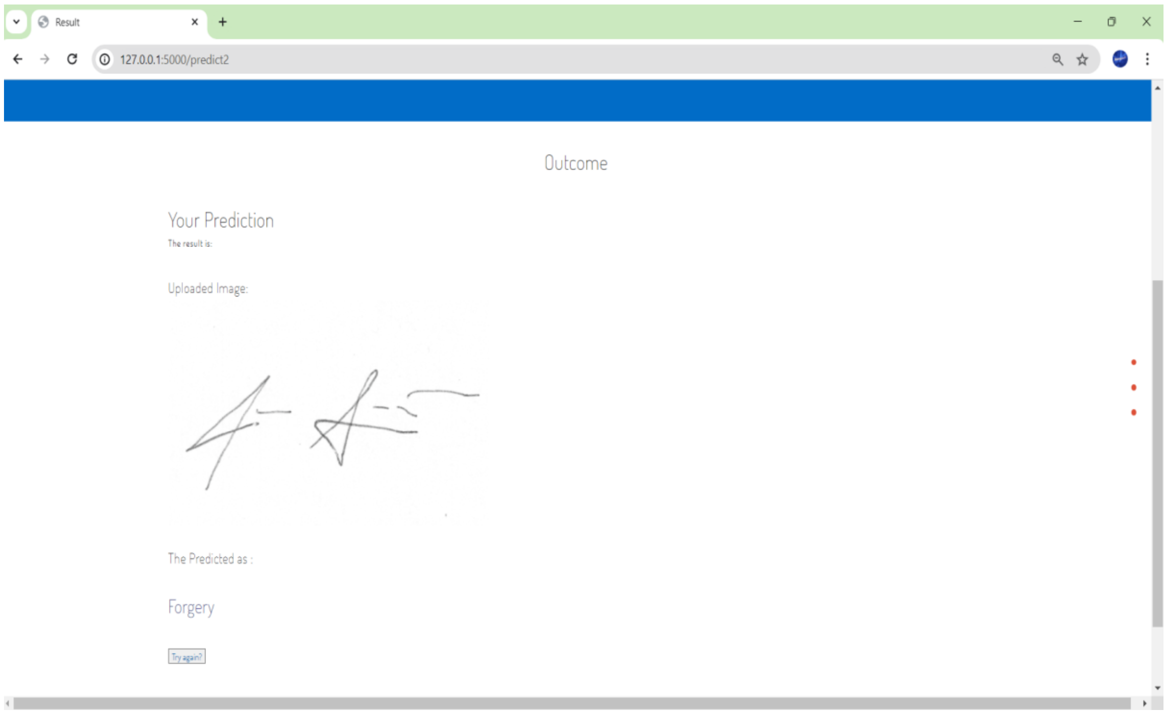
Uploaded Image:

The Predicted as :

Geniune

Screen 5.a: Predicting an Image as Genuine

Forgery


Result

127.0.0.1:5000/predict2

Outcome

Your Prediction

The result is:

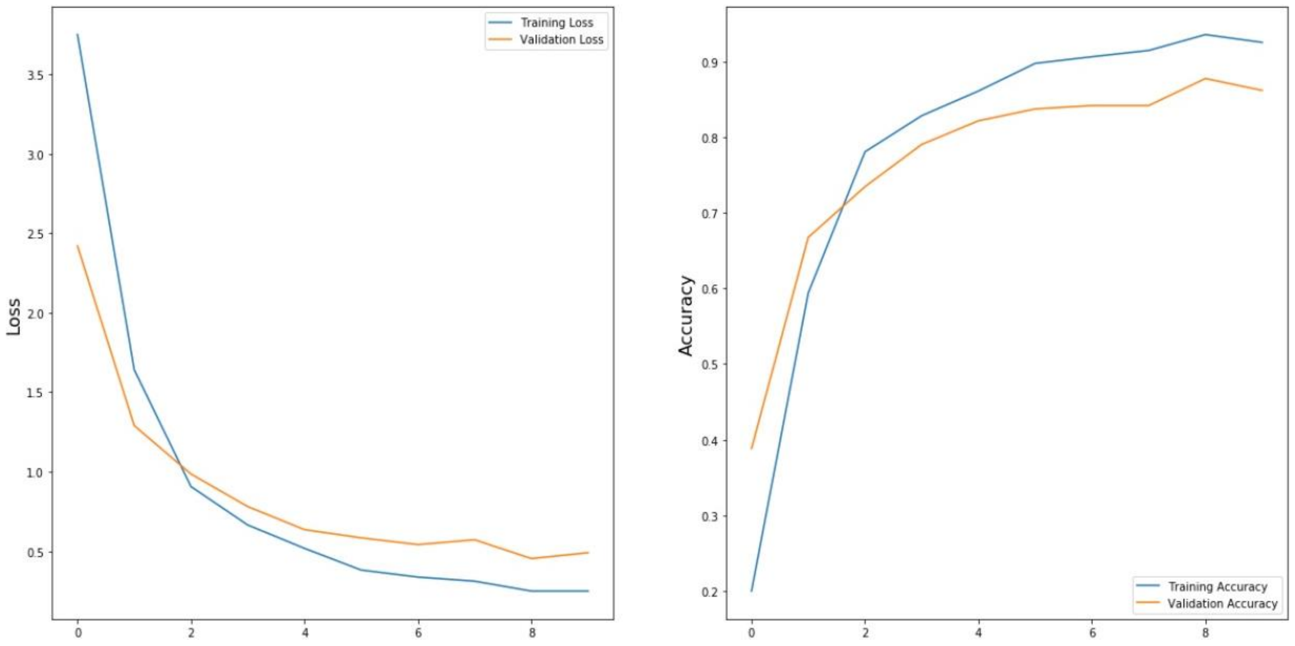
Uploaded Image:

The Predicted as :

Forgery

Screen 5.b: Predicting an Image as Forgery

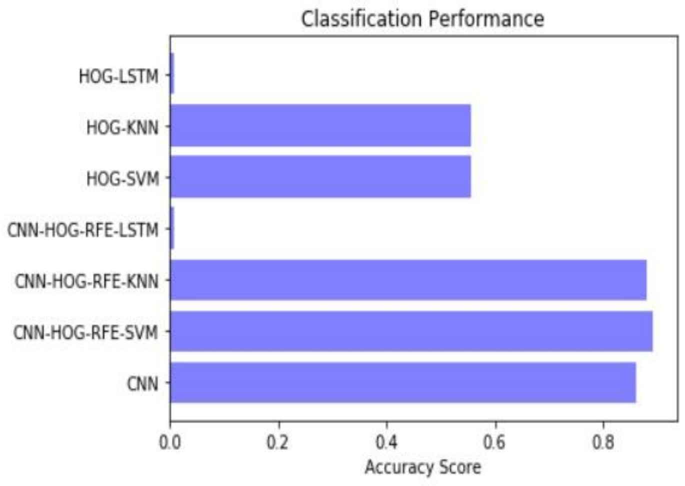
```
Epoch 1/10
132/132 [==============================] - ETA: 0s - loss: 1.7313 - accuracy: 0.9678 - f1_m: 0.9678 - precision_m: 0.9678 - rec
all_m: 0.9678
```



Screen 6.a: Comparison of Accuracy

Accuracy

```
In [141]: import matplotlib.pyplot as plt2
          plt2.barh(y_pos, accuracy, align='center', alpha=0.5,color='blue')
          plt2.yticks(y_pos, classifier)
          plt2.xlabel('Accuracy Score')
          plt2.title('Classification Performance')
          plt2.show()
```



Screen 6.b: Accuracy classification

10. CONCLUSION

The conclusion of the paper states that the proposed hybrid method for feature extraction in offline signature verification systems, which combines CNN and HOG techniques, followed by a feature selection algorithm. Three classifiers (LSTM, SVM, and KNN) were used for evaluation. With an good accuracy with the UTSig dataset and with the CEDAR dataset, the testing findings showed that our suggested model worked well in terms of performance and predictive capacity and achieved high accuracy in distinguishing between authentic and forged signatures, even for skilled forgeries. The paper highlights the significance of the feature extraction stage in offline signature verification systems and suggests that refining the feature extraction process could further improve the performance and prediction capability of these systems in the future.

11. REFERENCES

[1] F. M. Alsuhimat and F. S. Mohamad, ''Offline signature verification using long short-term memory and histogram orientation gradient,'' Bull. Elect. Eng. Inform., vol. 12, no. 1, pp. 283–292, 2023.

[2] K. Cpałka and M. Zalasiński, ''On-line signature verification using vertical signature partitioning,'' Expert Syst. Appl., vol. 41, no. 9, pp. 4170–4180, 2014.

[3] E. Griechisch, M. I. Malik, and M. Liwicki, ''Online signature verification based on Kolmogorov–Smirnov distribution distance,'' in Proc. 14th Int. Conf. Frontiers Handwriting Recognit., Sep. 2014, pp. 738–742.

[4] N. Sae-Bae and N. Memon, ''Online signature verification on mobile devices,'' IEEE Trans. Inf. Forensics Security, vol. 9, no. 6, pp. 933–947, Jun. 2014.

[5] M. Ajij, S. Pratihar, S. R. Nayak, T. Hanne, and D. S. Roy, ''Off-line signature verification using elementary combinations of directional codes from boundary pixels,'' Neural Comput. Appl., vol. 35, pp. 4939–4956, Mar. 2021, doi: 10.1007/s00521-021-05854-6.

[6] M. A. Ferrer, J. B. Alonso, and C. M. Travieso, ''Offline geometric parameters for automatic signature verification using fixed-point arithmetic,'' IEEE Trans. Pattern Anal. Mach. Intell., vol. 27, no. 6, pp. 993–997, Jun. 2005.

[7] Y. Guerbai, Y. Chibani, and B. Hadjadji, ''The effective use of the oneclass SVM classifier for handwritten signature verification based on writerindependent parameters,'' Pattern Recognit., vol. 48, no. 1, pp. 103–113, 2015.

[8] R. Larkins and M. Mayo, ''Adaptive feature thresholding for off-line signature verification,'' in Proc. 23rd Int. Conf. Image Vis. Comput. New Zealand, Nov. 2008, pp. 1–6.

[9] A. Q. Ansari, M. Hanmandlu, J. Kour, and A. K. Singh, ''Online signature verification using segment-level fuzzy modelling,'' IET Biometrics, vol. 3, no. 3, pp. 113–127, 2014.

[10] F. M. Alsuhimat and F. S. Mohamad, ''Offline signature verification using long short-term memory and histogram orientation gradient,'' Bull. Elect. Eng. Inform., vol. 12, no. 1, pp. 283–292, 2023.

[11] A. Q. Ansari, M. Hanmandlu, J. Kour, and A. K. Singh, ‘‘Online signature verification using segment-level fuzzy modelling,’’ IET Biometrics, vol. 3, no. 3, pp. 113–127, 2014.

[12] B. Kiran, S. Naz, and A. Rehman, ‘‘Biometric signature authentication using machine learning techniques: Current trends, challenges and opportunities,’’ Multimedia Tools Appl., vol. 79, no. 1, pp. 289–340, 2020.

[13] T.-A. Pham, H.-H. Le, and N.-T. Do, ‘‘Offline handwritten signature verification using local and global features,’’ Ann. Math. Artif. Intell., vol. 75, nos. 1–2, pp. 231–247, Oct. 2015.

[14] Z. ZulNarnain, M. S. Rahim, N. F. Ismail, and M. M. Arsad, ‘‘Triangular geometric feature for offline signature verification,’’ Int. J. Comput. Inf. Eng., vol. 10, no. 3, pp. 485–488, 2016.

[15] R. K. Bharathi and B. H. Shekar, ‘‘Off-line signature verification based on chain code histogram and support vector machine,’’ in Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI), Aug. 2013, pp. 2063–2068.

[16] R. Kumar, J. D. Sharma, and B. Chanda, ‘‘Writer-independent off-line signature verification using surroundedness feature,’’ Pattern Recognit. Lett., vol. 33, no. 3, pp. 301–308, Feb. 2012.

[17] N. Jiang, J. Xu, W. Yu, and S. Goto, ‘‘Gradient local binary patterns for human detection,’’ in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2013, pp. 978–981.

[18] N. M. Tahir, N. Adam, U. I. Bature, K. A. Abubakar, and I. Gambo, ‘‘Offline handwritten signature verification system: Artificial neural network approach,’’ Int. J. Intell. Syst. Appl., vol. 1, no. 1, pp. 45–57, 2021.

[19] A. B. Jagtap, D. D. Sawat, R. S. Hegadi, and R. S. Hegadi, ‘‘Verification of genuine and forged offline signatures using Siamese neural network (SNN),’’ Multimedia Tools Appl., vol. 79, nos. 47–48, pp. 35109–35123, Dec. 2020.

[20] B. Kiran, S. Naz, and A. Rehman, ‘‘Biometric signature authentication using machine learning techniques: Current trends, challenges and opportunities,’’ Multimedia Tools Appl., vol. 79, no. 1, pp. 289–340, 2020.

[21] M. Sharif, M. A. Khan, M. Faisal, M. Yasmin, and S. L. Fernandes, ‘‘A framework for offline signature verification system: Best features selection approach,’’ Pattern Recognit. Lett., vol. 139, pp. 50–59, Nov. 2020.

[22] N. Sharma, S. Gupta, and P. Metha, ‘‘A comprehensive study on offline signature verification,’’ in Proc. J. Phys., Conf., 2021, Art. no. 012044, doi: 10.1088/1742-6596/1969/1/012044.

[23] H. H. Kao and C. Y. Wen, ‘‘An offline signature verification and forgery detection method based on a single known sample and an explainable deep learning approach,’’ Appl. Sci., vol. 10, no. 1, p. 3716, 2020.

[24] M. K. Kalera, S. Srihari, and A. Xu, ‘‘Offline signature verification and identification using distance statistics,’’ Int. J. Pattern Recognit. Artif. Intell., vol. 18, no. 7, pp. 1339–1360, 2004.

[25] B. Kovari and H. Charaf, ‘‘A study on the consistency and significance of local features in off-line signature verification,’’ Pattern Recognit. Lett., vol. 34, no. 3, pp. 247–255, 2013.