Handwritten Digit Recognition for Enhanced Security - SAuth2.0

Student: Shivansh Mishra
Professor: Yang Long

***Abstract*—This paper discusses the development of a machine learning model - "SAuth2.0" to recognize handwritten digits as part of a multi-factor authentication system, enhancing security and user experience.**

I. Report Summary

A. *Introduction*

Current user authentication methods, such as passwords and biometrics, grapple with the challenge of achieving a harmonious balance between robust security and user privacy. Traditional methods are prone to vulnerabilities, while concerns regarding the privacy of biometric data persist. One-time passwords and app-based authentication hinge heavily on the security of the device.

In response to these challenges, there is a critical need for an advanced authentication solution—one that not only guarantees data protection and user privacy but also enhances overall security in authentication applications. The proposed solution attempts to addresses these concerns by incorporating Multi-Factor Authentication (MFA)-like security through a two-step verification process. The first step involves a password, and the second step introduces a user-entered handwritten digit, eliminating dependence on biometric or device-centric methods. This approach aims to deliver heightened security without compromising user privacy (1.

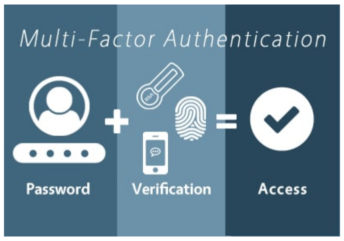


Fig. 1. MFA [1]

B. *Problem Statement, Inputs and Outputs*

*1) **Problem Statement:*** In user authentication applications, how to address existing vulnerabilities and anticipate and mitigate emerging threats, ensuring an optimal equilibrium between user privacy and the imperative of safeguarding sensitive data.

*2) **Objective:*** Develop a solution based upon a machine learning model to recognize handwritten digits as part of a multi-factor authentication system as a second step.

*3) **Input System:*** The system prompts users to handwrite a designated digit, such as '5,' employing either a stylus or touch input on a touchscreen or input device. Subsequently, the system captures this handwritten input as an image.

*4) **Output:*** The system adeptly identifies the handwritten digit and proceeds to either grant or deny permission accordingly. In the event of an inaccurate recognition, the user is prompted for a re-entry of the digit to ensure precise authentication.

Refer Fig 2 as a sample input and output.

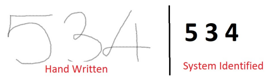


Fig. 2. Digit Recognition through ML

C. *Why Tree Based Learning -Random Forest & Logistic Regression Algorithms?*

After defining the problem and its inputs and outputs, the choice of algorithms becomes pivotal. In our context, utilizing Logistic Regression and Random Forest offers distinct advantages:

1. **Random Forest** - *Random Forest* excels in capturing intricate and non-linear patterns present in handwritten digit images, offering robustness through its ensemble nature. This complexity is well-suited for handling the diverse handwriting styles found in MNIST. Its ensemble approach enhances robustness and provides feature importance scores, offering insights into the significant characteristics of MNIST digits.

2. **Logistic Regression** - *Logistic Regression* provides simplicity and interpretability, making it a valuable baseline model. Its straightforwardness offers insights into linear relationships, allowing quick exploration of image data characteristics. Its clear feature impact assessment aids in understanding key pixel contributions for classification.

D. *Machine Learning Implementation*

1) **Algorithm Selection:**
   - Chose the Random Forest algorithm for its efficiency in complex classification tasks and resilience against overfitting.
2) **Hyperparameter Tuning:**
   - Fine-tuned the model by experimenting with hyperparameters to achieve optimal accuracy.
3) **Testing and Validation:**
   - Rigorously tested the model on the MNIST dataset, known for diverse handwriting styles.
   - Utilized another large dataset [3] for additional validation.

E. *Results, Limitations, and Lessons Learned*

*1) **Key Results**:* Through rigorous training on the MNIST dataset, the chosen model demonstrated exceptional performance, attaining an accuracy of 96.3% on the training data and an impressive 98.48% on an external Kaggle dataset [3]. See Fig12.These outcomes underscore the model's versatility in effectively handling diverse datasets, underscoring its potential for real-world applications.

*2) **Limitations**:*

1) **Universality Challenge:** - Enhance model robustness by incorporating diverse datasets, covering various writing styles, languages, and image qualities.
2) **Inclusivity Concern:** - Adapt to alternative input methods, like voice recognition, to ensure accessibility for users facing challenges such as physical disabilities or illiteracy.

*3) **Lessons Learned**:* This project underscores the pivotal role of parameter selection in influencing model accuracy, highlighting the imperative of strategic tuning. Moreover, insights into the significance of data quality, algorithmic versatility, and iterative testing have been garnered, providing valuable lessons for future endeavors in machine learning projects.

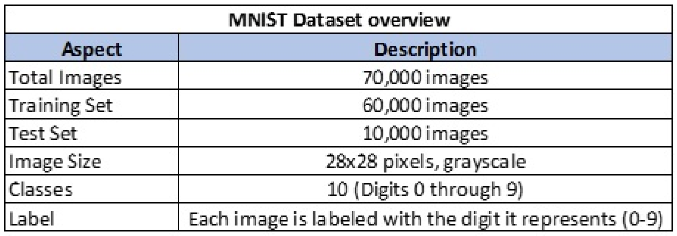
II. Data and Experimental Setup

A. *Dataset Details*

1. **MNIST Dataset:** - Directly imported into the Python program using `from tensorflow.keras.datasets import mnist`. This dataset was used to train the model. - More details on the Keras API can be found on the TensorFlow website [2].

2. **Kaggle Dataset:** - Downloaded locally and then referenced in the Python program. This dataset was used as an additional external dataset to validate the model.

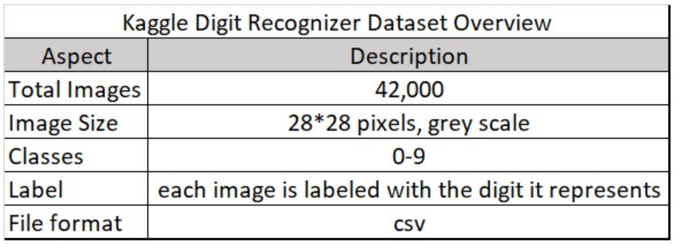
- Find more details on the data - MNIST Data (Fig 3) and Kaggle Data (Fig 4).



MNIST Dataset overview

Aspect	Description
Total Images	70,000 images
Training Set	60,000 images
Test Set	10,000 images
Image Size	28x28 pixels, grayscale
Classes	10 (Digits 0 through 9)
Label	Each image is labeled with the digit it represents (0-9)

Fig. 3. MNIST Dataset Details



Kaggle Digit Recognizer Dataset Overview

Aspect	Description
Total Images	42,000
Image Size	28*28 pixels, grey scale
Classes	0-9
Label	each image is labeled with the digit it represents
File format	csv

Fig. 4. Kaggle Dataset Details

B. *Sample digit images in MNIST dataset*

The sample digits of the MNIST dataset were generated using python program to get a visual sense and verification of the digits in the dataset.

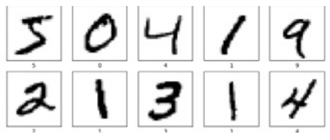


Fig. 5. Randomized images from the dataset

C. *Average digit images in MNIST dataset*

The Average digits of the MNIST dataset were generated using python program to understand for which digits the model would perform better.

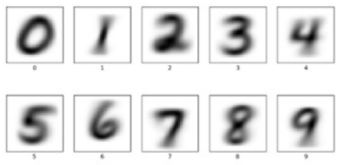


Fig. 6. Average digit images from MNIST dataset

*1) **Average digit images in MNIST dataset**:*

D. *Model Evaluation*

*1) **Algorithm Selection**:* In the quest to develop an efficient model for the MNIST dataset, two distinct machine learning algorithms from the course submodule were carefully selected for evaluation and comparison: the Random Forest Model and Logistic Regression. These algorithms were chosen based on their unique characteristics and proven track records in classification tasks.

1) **Random Forest Model (Proposed Algorithm for the Solution)**: This ensemble learning method, known for its high accuracy and robustness, constructs multiple decision trees and merges them for more reliable and accurate predictions. The Random Forest Model is particularly adept at handling high-dimensional data like images, making it an ideal candidate for handwritten digit recognition. Its ability to provide insights into feature importance and its resilience against overfitting are additional factors that motivated its selection.
2) **Logistic Regression (Baseline Algorithm)**: Serving as the baseline model, Logistic Regression is a simpler, yet effective, algorithm for binary and multiclass classification problems. Its straightforwardness and interpretability make it a standard choice for baseline models in many machine learning tasks. However, it might not capture the complexity present in image data as effectively as more sophisticated algorithms.

When both the algorithm were trained and tested in MNIST dataset, Random Forest Algorithm performed better than Logistic Regression as evidenced in Fig 7.

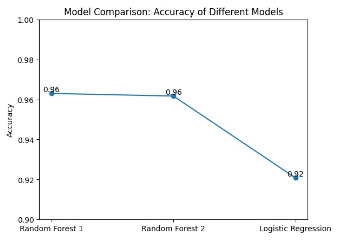
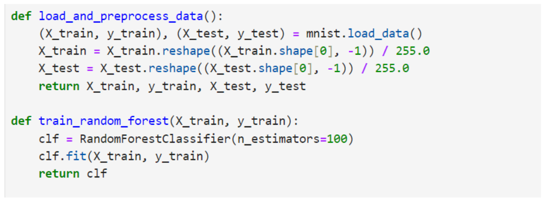


Fig. 7. Algorithm Comparison

This outcome was anticipated, given the inherent complexity and high dimensionality of the MNIST dataset, which aligns well with the strengths of the Random Forest algorithm. The comparison between these two models not only validated the superiority of the Random Forest Model for this particular task but also underscored the importance of algorithm selection in achieving high-performance results in machine learning projects.

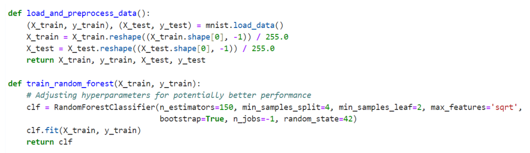
*2) **Model Optimization**:* After establishing that Random Forest Algorithm was a better option, a tuned Random Forest model was created to obtain better results with the Random forest algorithm. However, Model 1 still performed consistently better. Below are the optimization parameters in each Model 1 and Model 2.



```
def load_and_preprocess_data():
    (X_train, y_train), (X_test, y_test) = mnist.load_data()
    X_train = X_train.reshape((X_train.shape[0], -1)) / 255.0
    X_test = X_test.reshape((X_test.shape[0], -1)) / 255.0
    return X_train, y_train, X_test, y_test

def train_random_forest(X_train, y_train):
    clf = RandomForestClassifier(n_estimators=100)
    clf.fit(X_train, y_train)
    return clf
```

Fig. 8. Code of Random Forest Model 1



```
def load_and_preprocess_data():
    (X_train, y_train), (X_test, y_test) = mnist.load_data()
    X_train = X_train.reshape((X_train.shape[0], -1)) / 255.0
    X_test = X_test.reshape((X_test.shape[0], -1)) / 255.0
    return X_train, y_train, X_test, y_test

def train_random_forest(X_train, y_train):
    # Adjusting hyperparameters for potentially better performance
    clf = RandomForestClassifier(n_estimators=150, min_samples_split=4, min_samples_leaf=2, max_features='sqrt',
                                 bootstrap=True, n_jobs=-1, random_state=42)
    clf.fit(X_train, y_train)
    return clf
```

Fig. 9. Code of Random Forest Model 2

*3) **Comparison of Models**:* Utilizing the aforementioned parameters, the obtained results displayed striking similarity, albeit with a slight edge for Model 1. This nuanced distinction is visually apparent in the comparative graph (Fig 10).

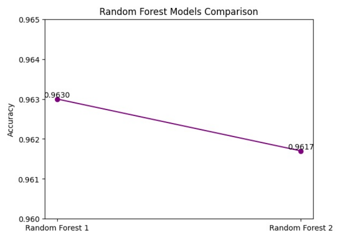


Fig. 10. Comparison of Model 1 vs Model 2

*4) **Explanation of Tuning Parameters**:* The tuning parameters in the Random Forest models (Model 1 and Model 2) were adjusted to enhance performance:

1) Increased Model Complexity: The number of trees ($n\_estimators$) was increased from 100 to 150, providing a more robust ensemble of decision trees.
2) Prevention of Overfitting: Parameters like $min\_samples\_split$, $min\_samples\_leaf$, and $max\_features$ were adjusted to introduce constraints on individual trees, preventing overfitting and promoting generalization.
3) Enhanced Parallel Processing and Reproducibility: The n_jobs parameter was set to -1, enabling parallel processing for efficient training. Additionally, a specific random seed ($random\_state = 42$) was set for result reproducibility.

These refinements sought to achieve a delicate equilibrium between model complexity and generalization, leading to enhanced performance. Notably, Model 1 exhibited consistent superiority over Model 2 underscoring the nuanced nature of model tuning and the necessity for meticulous testing to establish superiority.

E. Performance on Kaggle Dataset

The model demonstrated commendable performance on the Kaggle dataset (Fig 4). Fig 11 succinctly encapsulates the results obtained on the Kaggle dataset. Infract

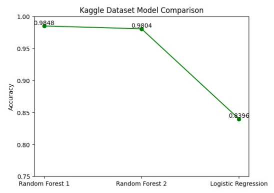


Fig. 11. Model Performance in Kaggle Dataset

III. Self-Evaluation Report

A. *Learnings from Lectures*

1) Key principles of Machine Learning for use in managing the dataset and building models.
2) Differences between supervised and unsupervised learning.
3) Different types of algorithms (Classification, Clustering, Regression, Dimensionality Reduction, etc.).
4) How to train, optimize, evaluate, and compare Machine Learning models.
5) Concepts of Underfitting and Overfitting, Cost functions, etc.

B. *Learnings from Coursework*

Below are my key learnings from Coursework.

1) Selection of the right algorithms based on the problem.
2) System requirements (CPU, Memory) for running AI/ML are much higher than regular programs.
3) TensorFlow, sklearn, Keras and Kaggle are some powerful tools and platform for AI.

C. *Challenges Faced in the Module*

Navigating through the module presented formidable challenges:

1) Model tuning after pinpointing the ideal algorithm, including strategies for resolution when the model deviates from expected performance.
2) Decision-making in data selection, collection, and preparation to suit the intricacies of the problem.

In overcoming these hurdles, I refereed external sources, notably consulting resources like [6] and [7]

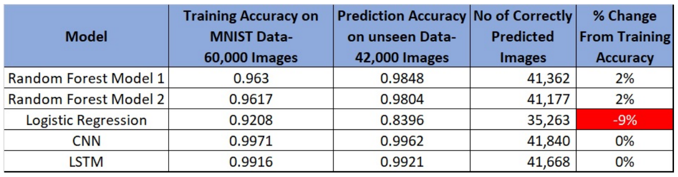
D. *What Would I do differently if I were to do it again*

1) Study already done work in the field by others and learn from best practices therein.
2) Invest more time in data collection and preparation.

E. *Unique Contributions or Novel Ideas*

To deepen my understanding of the topic and enhance my project, I researched and implemented two new algorithms, Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM), on the MNIST dataset. Following the training and testing phases, I proceeded to validate the performance of these models on Kaggle Data Set.

The insights garnered from my findings are intriguing, and I have meticulously documented the comparisons in Fig 12. Noteworthy is the observation that, according to four models either outperformed or maintained their performance, while the Logistic Regression model exhibited a decline. Further investigation is warranted to comprehend the factors contributing to this phenomenon.



Model	Training Accuracy on MNIST Data- 60,000 Images	Prediction Accuracy on unseen Data- 42,000 Images	No of Correctly Predicted Images	% Change From Training Accuracy
Random Forest Model 1	0.963	0.9848	41,362	2%
Random Forest Model 2	0.9617	0.9804	41,177	2%
Logistic Regression	0.9208	0.8396	35,263	-9%
CNN	0.9971	0.9962	41,840	0%
LSTM	0.9916	0.9921	41,668	0%

Fig. 12. Model Performance of CNN vs LSTM in Kaggle Dataset

IV. Conclusion

This project successfully showcased the application of machine learning in fortifying security measures through advanced handwriting recognition. Utilizing sophisticated algorithms, I piloted a system capable of discerning handwritten inputs, marking a significant improvement in integrating traditional security mechanisms with cutting-edge technology. The obtained results are promising, highlighting the potential of machine learning in security applications.

Yet, acknowledging the continuous journey towards perfection, there exists room for improvement. The current model, while substantial in accuracy, presents an opportunity for enhancement. To address this, I plan to work on, SAuth4.0, to elevate multi-factor authentication by integrating user-specific handwriting recognition. This feature ensures authentication codes are accepted only when matching the registered user's unique handwriting style, thus raising security to the next level.

Furthermore, the versatility of this solution extends to the development of analogous systems for other language scripts, such as Devanagiri for Hindi, a widely used Indian script. Research has already provided datasets for testing and training, as referenced in [4] and [5].

V. Appendix

Below are the Python files created are are included in submitted AI_ML.ipynb file.

1) 1_Train_Random_Forest_Model_1.py
2) 2_Train_Random_Forest_Model_2.py
3) 3_Train_Logistic_Regression_Model.py
4) 4_Test_All_Models_Kaggle_DataSet.py
5) 5_Plot_Comparision_Graphs.py
6) 6_Plot_MNIST_Dataset_Details.py
7) 7_CNN_LSTM_Model_Train_MNIST.py
8) 8_CNN_LSTM_Model_Predict_Kaggle.py
   To run 4_Test_All_Models_Kaggle_DataSet.py, it is necessary to download the csv files from the Kaggle dataset [3] as it contains the training and testing files.

VI. References

References

[1] Pirege Compliance. (2020). The Importance of Multifactor Authentication for Compliance and Safety. https://piregcompliance.com/authentication/the-importance-of-multifactor-authentication-for-compliance-and-safety/

[2] TensorFlow. (2023-06-08). Keras: The high-level API for TensorFlow. https://www.tensorflow.org/guide/keras

[3] Kaggle. (2021). Dataset for SAuth4.0. https://www.kaggle.com/competitions/digit-recognizer/data

[4] Kaggle. Dataset for SAuth4.0. https://www.kaggle.com/datasets/suvooo/hindi-character-recognition

[5] Medium. (2019-09-29). Dataset and information on Devangiri. https://medium.com/analytics-vidhya/cpar-hindi-digit-and-character-dataset-1347a7ff946

[6] LeeWayHertz. (2024). How to Choose the Right AI Model? https://www.leewayhertz.com/how-to-choose-an-ai-model/

[7] Towards Data Science. (2021-07-13). Considerations when choosing a machine learning model. https://towardsdatascience.com/considerations-when-choosing-a-machine-learning-model-aa31f52c27f3