Fraud Detection

Himansh
Arizona State University
hmudigon@asu.edu

Jainil Trivedi
Arizona State University
jtrived7@asu.edu

Sujith
Arizona State University
bpotinen@asu.edu

Vishrut Shah
Arizona State University
vshah80@asu.edu

Abstract

Fraud detection is a critical challenge in financial systems, particularly when it involves verifying identity documents at scale. Traditional solutions often rely on external AI services or manual review, introducing latency and security concerns. In this project, we propose a seamless integration of machine learning into database operations by embedding a custom Convolutional Neural Network (CNN) inside the Apache Spark ecosystem using User Defined Functions (UDFs). This approach allows image-based ID verification to be executed directly within SQL queries, enabling automated, high-throughput fraud checks without data movement or external dependencies. We explored and evaluated multiple technologies, including Velox, PostgreSQL, and EvaDB, but found them limited in handling complex ML models. Ultimately, Spark emerged as the most suitable platform. We tested both a pretrained EfficientNetB0 and a custom 3-layer CNN, finding the latter better suited for our domain-specific, small-scale dataset. Through comparative analysis of Spark vs. Pandas query execution and model performance, we demonstrate significant advantages in scalability and integration. Our solution bridges AI and databases in a practical and efficient manner, laying the foundation for real-time fraud prevention in data-driven applications.

1 Introduction

Fraudulent identity submissions are a growing concern in banking, e-commerce, and fintech applications, where verifying the authenticity of ID documents is a critical step in user onboarding and compliance. Traditional fraud detection systems often rely on third-party APIs or human-in-the-loop processes, introducing latency, increasing operational complexity, and creating potential security risks due to data movement across systems.

Our project aims to address these limitations by exploring a novel approach: embedding machine learning models directly into a database-like environment to perform real-time fraud detection as part of SQL queries. This eliminates the need for data export or separate AI inference pipelines. Specifically, we embed a Convolutional Neural Network (CNN) model within the Apache Spark ecosystem using User Defined Functions (UDFs), enabling end-to-end ID verification directly in the data processing layer.

This integration of AI into the database processing pipeline not only streamlines the verification process but also opens up possibilities for high-throughput, low-latency applications. By unifying analytics and ML inference under the same computational engine, we aim to reduce system overhead and improve the scalability of fraud prevention systems.

To position our work within the broader research landscape, we examined systems like Velox, PostgreSQL UDFs, and EvaDB, which attempt to bridge databases and AI. While these systems offer foundational capabilities, they often lack support for complex, multi-dimensional data like images or restrict the ML model types that can be used. This gap highlighted the need for a more flexible and scalable solution, ultimately motivating our Spark-based architecture.

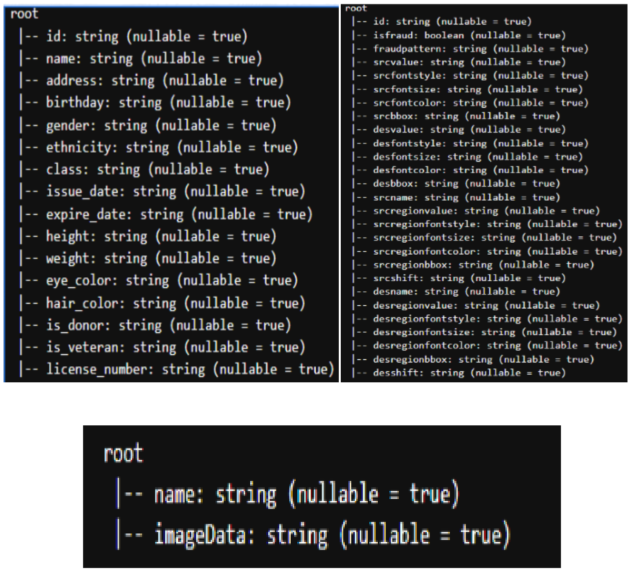
2 Dataset

The dataset used in this project is designed to simulate real-world ID verification scenarios, consisting of image data labeled for fraud detection. It includes three core fields:

- `idmeta`: Metadata about the ID, such as user ID, upload timestamp, or type.
- `idlabel`: The ground-truth label indicating whether the ID is **genuine** or **fraudulent**.
- `idimage`: The actual ID image encoded in a format suitable for ML processing.

These images serve as the primary input for our Convolutional Neural Network (CNN), which is trained to classify them as either valid or fraudulent based on visual patterns.

The dataset is relatively small, making it ideal for experimenting with lightweight model architectures and validating performance across query engines like Pandas and Spark. Despite its size, it captures the essential characteristics of fraud detection use cases where model adaptability and fast inference are crucial.



```
root
 |-- id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- address: string (nullable = true)
 |-- birthday: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- ethnicity: string (nullable = true)
 |-- class: string (nullable = true)
 |-- issue_date: string (nullable = true)
 |-- expire_date: string (nullable = true)
 |-- height: string (nullable = true)
 |-- weight: string (nullable = true)
 |-- eye_color: string (nullable = true)
 |-- hair_color: string (nullable = true)
 |-- is_donor: string (nullable = true)
 |-- is_veteran: string (nullable = true)
 |-- license_number: string (nullable = true)
```

```
root
 |-- id: string (nullable = true)
 |-- isfraud: boolean (nullable = true)
 |-- fraudpattern: string (nullable = true)
 |-- srcvalue: string (nullable = true)
 |-- srcfontstyle: string (nullable = true)
 |-- srcfontsize: string (nullable = true)
 |-- srcfontcolor: string (nullable = true)
 |-- srcbbox: string (nullable = true)
 |-- desvalue: string (nullable = true)
 |-- desfontstyle: string (nullable = true)
 |-- desfontsize: string (nullable = true)
 |-- desfontcolor: string (nullable = true)
 |-- desbbox: string (nullable = true)
 |-- srcname: string (nullable = true)
 |-- srcregionvalue: string (nullable = true)
 |-- srcregionfontstyle: string (nullable = true)
 |-- srcregionfontsize: string (nullable = true)
 |-- srcregionfontcolor: string (nullable = true)
 |-- srcregionbbox: string (nullable = true)
 |-- srcshift: string (nullable = true)
 |-- desname: string (nullable = true)
 |-- desregionvalue: string (nullable = true)
 |-- desregionfontstyle: string (nullable = true)
 |-- desregionfontsize: string (nullable = true)
 |-- desregionfontcolor: string (nullable = true)
 |-- desregionbbox: string (nullable = true)
 |-- desshift: string (nullable = true)
```

```
root
 |-- name: string (nullable = true)
 |-- imageData: string (nullable = true)
```

A visual showing the three columns (idmeta, idlabel, idimage) along with example entries or types.

3 Environment Setup

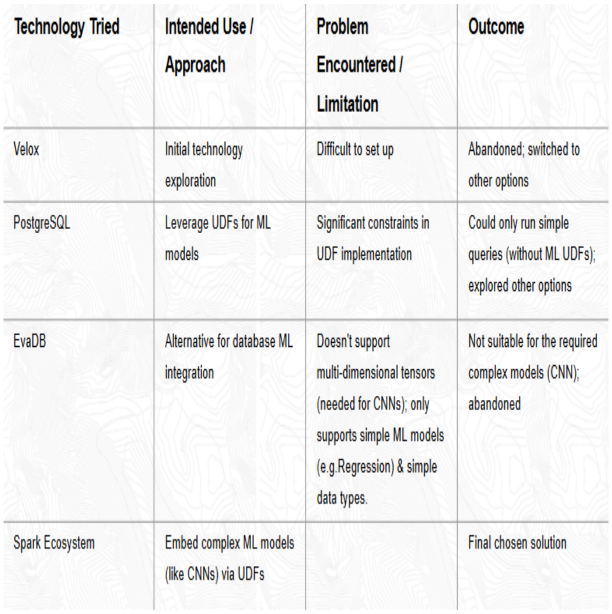
To integrate deep learning inference within a data processing pipeline, we experimented with multiple technologies that claimed to support ML workloads alongside structured queries. Each system presented its own set of advantages and limitations, which shaped the evolution of our implementation strategy.

We began with **Velox**, a high-performance unified execution engine designed for ML and analytical workloads. While conceptually promising, Velox proved extremely difficult to set up due to incomplete documentation and complex build dependencies. We abandoned it early in the process.

Next, we explored **PostgreSQL**, leveraging its support for User Defined Functions (UDFs) to embed ML models. However, PostgreSQL's UDF environment has significant constraints: limited language support, restricted memory management, and no native support for multi-dimensional tensor inputs such as images. As a result, we were only able to execute basic queries without ML integration.

We then turned to **EvaDB**, a database system designed to support ML inference. Although EvaDB simplifies embedding models like XGBoost or logistic regression, it does not support multi-dimensional tensors or deep learning models like CNNs. This made it unsuitable for our use case.

Finally, we adopted the **Apache Spark ecosystem**, which allowed us to embed a custom CNN model using PySpark UDFs. Spark's scalability and Python compatibility made it the most effective platform for our requirements.



Technology Tried	Intended Use / Approach	Problem Encountered / Limitation	Outcome
Velox	Initial technology exploration	Difficult to set up	Abandoned; switched to other options
PostgreSQL	Leverage UDFs for ML models	Significant constraints in UDF implementation	Could only run simple queries (without ML UDFs); explored other options
EvaDB	Alternative for database ML integration	Doesn't support multi-dimensional tensors (needed for CNNs); only supports simple ML models (e.g.Regression) & simple data types.	Not suitable for the required complex models (CNN); abandoned
Spark Ecosystem	Embed complex ML models (like CNNs) via UDFs		Final chosen solution

A table comparing Velox, PostgreSQL, EvaDB, and Spark across columns like "Intended Use," "Problem Faced," and "Outcome."

4 Methodology

Our approach centers around embedding a deep learning model within the Apache Spark environment using User Defined Functions (UDFs). This allows image-based fraud detection to run as part of Spark SQL queries—making AI inference seamless, scalable, and parallelizable.

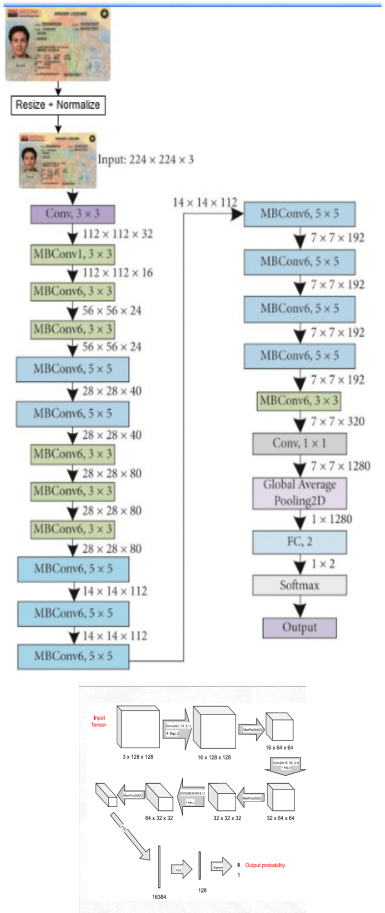
4.1 Model Architecture

Initially, we experimented with **EfficientNetB0**, a lightweight yet powerful convolutional neural network known for its performance on standard image classification tasks. However, it showed **limited generalization** to our domain-specific ID image dataset and proved **difficult to fine-tune** due to its complexity and overfitting tendencies on the small dataset.

To address this, we designed a **custom CNN architecture** consisting of:

- 3 convolutional layers with ReLU activations
- MaxPooling layers for dimensionality reduction
- Fully connected layers leading to a binary classification output

This simpler architecture provided better adaptability to our dataset and allowed faster, more stable training cycles.

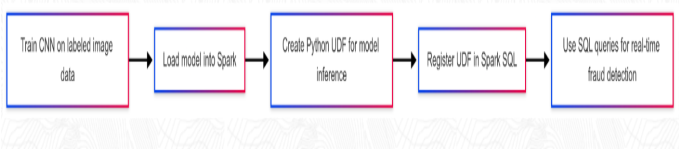


A picture showing EfficientNetB0 and Custom CNN based on accuracy, generalization, fine-tuning, and complexity.

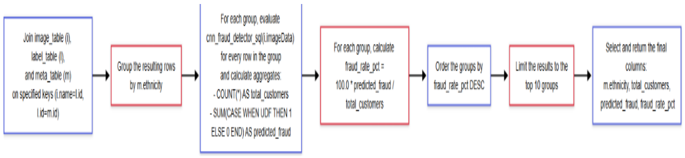
4.2 Integration with Spark using UDFs

We used **PySpark UDFs** to embed our trained CNN model directly into SQL-like Spark queries. The UDF accepts base64-encoded image data as input, decodes it, preprocesses it into a tensor, and feeds it into the CNN model for prediction. This makes the fraud detection process fully compatible with structured batch or streaming workflows.

The UDF was registered in Spark and could be invoked like any other SQL function, allowing us to combine deep learning inference with filtering, joining, and aggregation operations natively in Spark.


Train CNN on labeled image data
Load model into Spark
Create Python UDF for model inference
Register UDF in Spark SQL
Use SQL queries for real-time fraud detection

Shows the end-to-end system: Data ingestion → Spark SQL Query → CNN UDF → Prediction Output.


Join image_table (i), label_table (l), and meta_table (m) on specified keys (i.name=l.id, l.id=m.id)
Group the resulting rows by m.ethnicity
For each group, evaluate cnn_fraud_detector_sql(i.imageData) for every row in the group and calculate aggregates: - COUNT(*) AS total_customers - SUM(CASE WHEN UDF THEN 1 ELSE 0 END) AS predicted_fraud
For each group, calculate fraud_rate_pct = 100.0 * predicted_fraud / total_customers
Order the groups by fraud_rate_pct DESC
Limit the results to the top 10 groups
Select and return the final columns: m.ethnicity, total_customers, predicted_fraud, fraud_rate_pct

Illustrates SQL query invoking a UDF, which loads model, preprocesses image, runs prediction, and returns result.

5 Results

To evaluate the effectiveness of our fraud detection system, we conducted experiments measuring both **model accuracy** and **query performance** across different platforms. This section presents our findings from comparing models, analyzing Spark vs Pandas query execution, and understanding how system scalability is affected by dataset size.

5.1 Model Evaluation

We first tested a pretrained **EfficientNetB0** model on our dataset and observed a validation accuracy of approximately **81%**. While the model performed decently, it showed signs of overfitting and struggled to generalize to subtle ID fraud patterns.

In contrast, our **custom CNN** achieved **comparable accuracy** with a **simpler architecture** and significantly **better training stability**. This validated our hypothesis that a lightweight, domain-tuned model is more suitable for small and specific datasets like ours.

5.2 Query Performance: Spark vs Pandas

To benchmark the performance of ML-driven queries across systems, we compared **PySpark** and **Pandas** implementations for four representative fraud detection queries. These queries simulate operations like bulk ID verification, fraud rate computation, and classification filtering. Each query was tested across multiple dataset scales (small, medium, large), and execution time was measured.

5.3 Reason: Why Spark is faster

1. **Distributed Processing:** Spark operates in a distributed fashion across multiple cores or machines (16 in our case), enabling parallel execution of tasks. In contrast, Pandas runs on a single machine and processes data sequentially, leading to slower execution for large-scale operations.
2. **Optimized Computation and Memory Usage:** Spark uses in-memory computation, which minimizes disk I/O and speeds up data processing. Additionally, it decodes data such as base64 in batches. Pandas, however, performs such operations sequentially and loads the entire dataset into memory, often leaving insufficient space for further processing.
3. **SQL Integration:** Spark supports SQL natively, allowing for declarative querying and built-in optimizations that improve performance. Pandas requires procedural coding for similar tasks, which can be less efficient and harder to optimize.

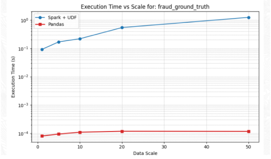
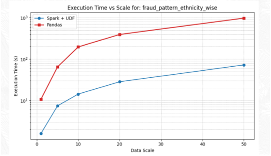
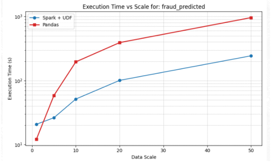
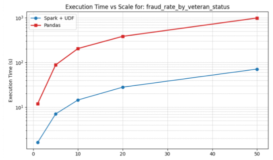
<table>
<tr><th>Query</th><th>Performance</th></tr>
<tr><td><b>Fraud Ground Truth</b></td><td>
</td></tr>
<tr><td><b>Fraud Pattern Ethnicity wise</b></td><td>
</td></tr>
<tr><td><b>Fraud Predicted</b></td><td>
</td></tr>
<tr><td><b>Fraud Rate by Vetran Status</b></td><td>
</td></tr>
</table>

Table 1: Spark vs Pandas Query Comparison Table

5.4 Observations

- **Spark queries scaled significantly better** than Pandas as dataset size increased.
- While Pandas showed slightly lower latency for small datasets, it degraded rapidly beyond medium scale.
- Spark UDFs ran seamlessly inside SQL-like commands and benefited from parallel execution.
- The **custom CNN's integration with Spark UDF** achieved real-time fraud detection capability on large datasets without needing external inference servers.

These results confirm that Spark not only enables effective integration of machine learning models but also provides superior scalability and performance for practical data processing scenarios.

6 Skills/Knowledge Learned

Working on this project offered valuable hands-on experience in bridging machine learning and database systems—an emerging intersection in modern data engineering.

6.1 Key Lessons

- **Understanding System Limitations**: We learned firsthand how different database systems like Velox, PostgreSQL, and EvaDB handle (or fail to handle) machine learning integration. Each tool taught us something new about constraints, from UDF environment limitations to lack of tensor support.
- **Model-Data Compatibility**: We saw how even strong models like EfficientNetB0 can fail to generalize when applied to narrow domain datasets, reinforcing the importance of designing lightweight, problem-specific architectures.
- **Working with Spark UDFs**: Embedding a deep learning model in Spark taught us how to manipulate data at scale while retaining SQL-like accessibility. This deepened our understanding of PySpark, serialization, and the internal workings of UDF execution.
- **Performance Engineering**: Comparing Pandas vs Spark forced us to think critically about scalability, execution bottlenecks, and data movement—skills essential for building production-grade data pipelines.
- **Team Collaboration & Debugging**: Finally, this project helped us develop collaboration skills in troubleshooting complex, multi-layered environments where ML and distributed systems interact.

7 Future Work

While our current implementation successfully integrates a custom CNN into Spark for real-time fraud detection, there are several directions for future improvements and extensions.

7.1 Potential Enhancements

- **Model Generalization**: Our current model is tailored to a small, domain-specific dataset. Expanding the dataset and retraining the model on more diverse ID types (e.g., passports, driver's licenses, global formats) would improve robustness and generalization to real-world use cases.
- **Multi-modal Fraud Detection**: Beyond images, future systems could incorporate additional metadata (e.g., geolocation, device info, user behavior) to build a multi-input fraud scoring system using ensemble models.
- **System Portability**: While Spark proved effective, integrating similar UDF-based inference in systems like **DuckDB**, **PostgreSQL with PL/Python**, or enhanced **EvaDB** support could make the solution accessible to teams without big data infrastructure.
- **Streaming Integration**: Currently, our pipeline operates in batch mode. Adapting it for streaming environments using **Spark Structured Streaming** would allow near-instant fraud checks in production APIs.
- **Explainability Tools**: Introducing explainable AI (XAI) techniques like Grad-CAM or attention maps could help human reviewers understand why a particular ID was flagged as fraudulent, increasing trust in the system.

8 Conclusion

This project demonstrates the feasibility and effectiveness of integrating deep learning models directly into data processing pipelines for fraud detection. By embedding a custom Convolutional Neural Network (CNN) into the Apache Spark ecosystem via User Defined Functions (UDFs), we enabled real-time ID verification to operate natively within

SQL queries—without requiring external inference services or manual intervention.

Throughout our exploration, we evaluated multiple systems such as Velox, PostgreSQL, and EvaDB. While each offered partial support for AI integration, none provided the flexibility and tensor compatibility needed for CNN-based image classification at scale. Spark emerged as the ideal platform, enabling seamless integration of ML logic, efficient parallel execution, and SQL-level query accessibility.

Our results confirm that this architecture can scale to large datasets, outperform traditional Python-based tools like Pandas in runtime, and deliver high prediction accuracy using lightweight models. The project not only offered technical insights into system limitations and performance engineering, but also highlighted the growing importance of unifying AI and data infrastructure.

In doing so, we contribute to a broader trend in data science: moving intelligence closer to the data—enabling smarter, faster, and more secure operations within modern analytics systems.

References

by Examples, S. 2025. Pyspark udf (user defined function). https://sparkbyexamples.com/pyspark/pyspark-udf-user-defined-function/. Accessed: 2025-05-03.

Group, G. T. D. 2025a. Evadb - ai-powered database system. https://github.com/georgia-tech-db/evadb. Accessed: 2025-05-03.

Group, P. G. D. 2025b. Postgresql official site. https://www.postgresql.org/. Accessed: 2025-05-03.

Group, P. G. D. 2025c. User-defined functions in postgresql. https://www.postgresql.org/docs/current/xfunc.html. Accessed: 2025-05-03.

PostgresML. 2025. Postgresml - machine learning in postgres. https://postgresml.org/. Accessed: 2025-05-03.

PyTorch. 2025. Pytorch installation guide. https://pytorch.org/get-started/locally/#windows-python. Accessed: 2025-05-03.

Spark, A. 2025. Apache spark python api docs. https://spark.apache.org/docs/latest/api/python/index.html. Accessed: 2025-05-03.

TensorFlow. 2025. Efficientnetb0 api documentation. https://www.tensorflow.org/api_docs/python/tf/keras/applications/EfficientNetB0. Accessed: 2025-05-03.

(Spark 2025; Group 2025a; 2025b; 2025c; by Examples 2025; PyTorch 2025; TensorFlow 2025; PostgresML 2025)