WORKSHOP #2
KAGGLE SYSTEMS DESIGNS – NFL IMPACT DETECTION

INTEGRANTS
NAHIN JOSE PEÑARANDA MEJIA (njpenarandam@udistrital.edu.co)
NICOLAS FELIPE PULIDO SUAREZ (nfpulidos@udistrital.edu.co)
ANDERSON DAVID ARENAS GUTIERREZ (adarenasg@udistrital.edu.co)

SYSTEMS ANALYSIS & DESIGN
UNIVERSIDAD DISTRITAL FRANCISCO JOSE DE CALDAS
MAY – 2025
BOGOTA – COLOMBIA

1. Summary of Systems Analysis Findings:

The systems analysis identified several key challenges that need to be addressed in the design. First, the data quality varies, with sensor inaccuracies and low video frame rates making it harder to extract reliable information. The system also should process data in real time, which is demanding on computational resources. Synchronizing sensor and video data is another challenge, as both must be perfectly aligned in time to ensure accuracy. Additionally, rare events, like significant player contacts, create imbalanced data that is difficult for models to handle.

The dataset is rich and includes sensor data, video feeds, and contact labels, but this complexity brings problems. Tracking players in crowded scenes or during fast movements is difficult, and environmental factors, like lighting or obstructions, can reduce video quality. Ground truth labels are essential for training the system, but errors or inconsistencies in these labels can cause issues later in the process.

The system also faces challenges related to combining different types of data. Sensor and video data need to be merged carefully to avoid errors, and the system's accuracy depends on fine-tuned thresholds for decision-making. Random factors, like players overlapping or external conditions, make the system less predictable.

To solve these problems, the design should include tools for efficient real-time data processing, techniques for combining sensor and video data, and models that can handle imbalanced data and unexpected situations. This will make the system more reliable, scalable, and easy to maintain.

2. System Requirements

Based on the analysis, the system must meet the following requirements:

- **Functional Requirements:**
  - Ingest and process video and sensor data from NFL games.
  - Detect and track players in video footage accurately.
  - Extract features from video and sensor data relevant to contact events.
  - Predict contact events with high precision and recall.
  - Deploy the model for inference on new data, supporting both batch and potential real-time processing.
- **Non-Functional Requirements:**
  - **Scalability:** Handle data from multiple games or seasons.
  - **Performance:** Achieve low-latency processing for near-real-time applications.

    - **Reliability:** Ensure consistent predictions across varied game conditions.
    - **Maintainability:** Facilitate updates to models and pipelines.
    - **Security:** Protect player data privacy during processing and storage.
- **User-Centric Needs:**
    - **Interpretability:** Provide insights into why contacts are detected (e.g., feature importance).
    - **Ease of Use:** Offer intuitive interfaces for analysts or coaches.
    - **Robustness:** Perform reliably under diverse game scenarios.

3. High-Level Architecture

The system is designed as a modular pipeline, ensuring flexibility and scalability. The architecture includes the following components:

- **Data Ingestion:** Collects video and sensor data from multiple sources.
- **Data Preprocessing:** Synchronizes, cleans, and prepares data for analysis.
- **Player Detection and Tracking:** Identifies and tracks players in video footage.
- **Feature Extraction:** Extracts relevant features from video and sensor data.
- **Contact Detection:** Uses machine learning to predict contact events.
- **Deployment:** Serves the model for inference on new data.
- **Monitoring:** Tracks system performance and data quality, triggering updates as needed.

Architecture Diagram (Textual Description):

```
[Data Sources: Video, Sensors, Metadata] → [Ingestion] →
[Preprocessing] → [Detection & Tracking] → [Feature Extraction] →
[Contact Detection] → [Deployment] → [Monitoring]
```

Feedback from the Monitoring module loops back to the Contact Detection module for model retraining.

Systems Engineering Principles:

- **Modularity:** Independent components for easy modification.
- **Efficiency:** Lightweight tools to minimize resource usage.
- **Simplicity:** Jupyter notebooks for user-friendly execution.

4. Addressing Sensitivity and Chaos

To manage chaotic player interactions and sensitive elements:

- **Chaos Mitigation:**
    - **Temporal Modeling:** Incorporate time-series or sequence-based models to capture dynamic player movements over multiple frames.
    - **Ensemble Methods:** Combine predictions from multiple models to stabilize outputs against unpredictable variations.
    - **Data Augmentation:** Simulate varied player interactions to train models on diverse scenarios.
- **Sensitivity Management:**
    - **Robust Detection:** Use advanced object detection models (e.g., YOLOv5) to handle occlusions and noise in video data.
    - **Regularization:** Apply techniques like dropout or weight decay to prevent overfitting to noisy sensor data.
    - **Uncertainty Estimation:** Incorporate probabilistic outputs to quantify confidence in contact predictions.
    - **Monitoring Routines:** Continuously track data distributions and model performance to detect and address drifts or anomalies.

5. Technical Stack and Implementation Sketch

To accommodate less powerful hardware than the winner, the technical stack is focusing on CPU-friendly tools and local execution:

Category	Tool/Library	Justification
Programming Language	Python	Extensive data science ecosystem, widely accessible.
Data Processing	Pandas, NumPy	Efficient for sensor data manipulation, low memory footprint.
Video Processing	OpenCV	Lightweight for frame extraction and tracking, CPU-optimized.
Machine Learning	TensorFlow (MobileNet SSD)	Pre-trained, lightweight object detection model suitable for CPU.

Machine Learning	Scikit-learn (RandomForest)	Efficient for contact classification, minimal resource requirements.
Execution	Jupyter Notebooks	Interactive, user-friendly for workshops, supports step-by-step execution.
Version Control	Git	Standard for code versioning, lightweight and essential.

Implementation Plan

- **Environment Setup:**
  - Install Python 3.8+ and create a virtual environment using venv.
  - Install libraries
  - Ensure compatibility with CPU-only execution.
- **Data Preparation:**
  - Select a small subset of the competition dataset (e.g., a few video clips and sensor data).
  - Use Pandas to synchronize and clean data, reducing memory usage by processing in batches.
- **Player Detection:**
  - Load a pre-trained MobileNet SSD model from TensorFlow's Object Detection API (TensorFlow Models).
  - Apply to video frames at reduced resolution (e.g., 640x480) or lower frame rate (e.g., 1 FPS) to detect players.
- **Tracking:**
  - Use OpenCV's CSRT tracker (OpenCV Tracking) to track detected players across frames, minimizing computational load.
- **Feature Extraction:**
  - Compute features from detection (player positions, bounding box sizes) and tracking (velocities, trajectories).
  - Extract sensor data features (e.g., acceleration, proximity) using Pandas.
- **Contact Detection:**
  - Train a RandomForestClassifier from Scikit-learn (Scikit-learn) on extracted features to predict contact events.
  - Use cross-validation to ensure robustness with limited data.
- **Execution and Monitoring:**
  - Run the pipeline in a Jupyter notebook, processing data sequentially.

    - Implement basic logging to track performance metrics (e.g., accuracy, runtime) and errors.

Design Patterns

- **Observer Pattern:** Basic logging to monitor performance without complex orchestration.
- **Factory Pattern:** Allow flexible model selection (e.g., RandomForest vs. LogisticRegression).

References

- **https://www.kaggle.com/competitions/nfl-player-contact-detection/team**
- https://github.com/nvnnghia/1st_place_kaggle_player_contact_detection/tree/main
- https://github.com/ItzNxhin/SAD---Nahin-Nicolas-and-Anderson/tree/main/Workshop1%20-%20Kaggle