CHESS LENS

A PROJECT REPORT

submitted by

"BSB64"
(Soham Bhattacharya, M.Sc. Big Data Analytics, Semester 2, B2430059,
Ronak Sarkar, M.Sc. Big Data Analytics, Semester 2, B2430054 &
Darpan Bhattacharya, M.Sc. Big Data Analytics, Semester 2, B2430044)

as a part of Computer Vision Course (CS342) of
Master of Science in Big Data Analytics program

Department of Computer Science
Ramakrishna Mission Vivekananda Educational and Research Institute
Belur, Howrah - 711202

April 29, 2025

ACKNOWLEDGEMENT

We would like to thank our course instructor, Br. Tamal Mj, for providing us with the opportunity to work on such an interesting project. We also appreciate the guidance and the occasional pointers that he gave us along the way for this project.

Soham Bhattacharya (MSc BDA, Sem 2, B2430059),
Ronak Sarkar (MSc BDA, Sem 2, B2430054) &
Darpan Bhattacharya (MSc BDA, Sem 2, B2430044)

CONTENTS

LIST OF FIGURES

Chapter 1
INTRODUCTION

BACKGROUND

Chess is a timeless and universally recognized board game, celebrated for its strategic depth and intellectual challenge, played by millions across the globe in settings ranging from casual home games to high-stakes international championships. Its global appeal has led to the development of technologies aimed at enhancing the experience of players and spectators alike. The Digital Game Technology (DGT) chessboard represents the current gold standard for digitizing live chess matches, offering seamless real-time tracking of moves and enabling broadcasters to share game progress with audiences worldwide. However, the prohibitive cost of DGT boards, often priced in the range of several hundred dollars, poses a significant barrier for smaller entities such as community chess clubs, schools, or universities with limited budgets. In the absence of such technology, event organizers frequently resort to manual simulation, where individuals manually input moves into a digital system to project the game state. This process is not only labor-intensive but also highly susceptible to human error, especially in the fast-paced environment of live tournaments where accuracy and speed are critical. These challenges highlight the need for a more accessible and efficient solution to digitize chess games, particularly for organizations operating under resource constraints.

MOTIVATION

The inspiration for the ChessLens project emerged from a practical challenge faced during the “Better Call Tal” chess event, a highlight of the 2025 tech fest organized by the Computer Science department of RKMVERI. The event drew a large and enthusiastic crowd, necessitating the projection of live chess games onto screens to ensure all spectators could follow the action. Without access to costly DGT boards, projecting the games during the event relied on manual simulation to replicate moves on a digital platform. This approach proved to be cumbersome, time-consuming, and prone to mistakes, which compromised the quality of the viewing experience and added significant pressure to the event’s operations. Following the event, a pivotal discussion with our course instructor, Br. Tamal mj, sparked the idea of harnessing computer vision to create an automated, cost-effective alternative to DGT boards. Motivated by this vision, we set out to develop ChessLens, a system designed to use a single camera and advanced computer vision techniques to digitize chess games in real time. Our goal was to create a solution that not only overcomes the financial and logistical barriers of existing technologies but also enhances the efficiency and accessibility of chess event management, making it feasible for institutions like RKMVERI and similar organizations to host high-quality chess tournaments.

OBJECTIVE

The **ChessLens** project aims to develop a cost-effective, computer vision-based system to digitize physical chess games in real-time, offering an accessible alternative to costly Digital Game Technology (DGT) boards. The specific objectives are:

1. **Develop a Real-Time Digitization System**: Create a system that uses a single camera, such as a mobile phone camera, to capture and digitize a physical chessboard's state during a live game, adhering to standard FIDE rules.

2. **Accurate Chessboard Detection**: Implement computer vision techniques, including Harris corner detection and Hough Line Transform (traditional approach) or homography (modern approach), to accurately detect the chessboard and segment it into 64 squares and identifying whether each square is occupied or empty.

3. **Track Moves and Update Board State**: Enable the system to track moves after each half-move (white or black) by comparing board states, handling standard moves, captures, and special cases like pawn promotions, castling, and en passant.

4. **Support Rapid and Classical Games**: Ensure reliable performance for rapid and classical chess formats, with an aspirational goal of supporting blitz games, addressing challenges such as lighting variations, perspective distortions, and player interactions (e.g., hand occlusions).

5. **Create a User-Friendly Interface**: Develop a graphical user interface (GUI) to visualize the digitized chessboard, facilitating real-time updates for spectators and event organizers.

6. **Enhance Event Management**: Streamline chess event management, such as the *Better Call Tal* tournament at Ramakrishna Mission Vivekananda Educational and Research Institute (RKMVERI), by reducing reliance on error-prone manual simulation and eliminating the need for expensive DGT boards.

Chapter 2
LITERATURE REVIEW

The application of computer vision to digitize chess games has garnered significant attention due to its potential in real-time game tracking, tournament broadcasting, and autonomous chess systems. Achieving accurate chessboard and piece recognition under diverse conditions remains a complex challenge, necessitating robust and efficient methodologies. This literature review examines three pivotal studies relevant to our ChessLens project: Liu et al. (2015), which focuses on automatic chessboard corner detection; Bugarin (2024), which presents a real-time chess vision system; and Shan and Ju (undated, referenced as Shan & Ju), which proposes a convolutional neural network (CNN)-based approach for converting physical chessboard images to Forsyth-Edwards Notation (FEN). These works provide foundational techniques and insights that guide our development of a cost-effective, computer vision-based system for digitizing chess games.

Liu et al. (2015): Automatic Chessboard Corner Detection Method

Liu et al. (2015) introduced an algorithm for automatic chessboard corner detection, published in *IET Image Processing*, aimed at enhancing camera calibration and chessboard localization. The method focuses on detecting X-corners (inner corners at black-and-white square intersections) through four steps: initial corner detection, fake corner elimination, corner sorting, and sub-pixel localization. Hessian corner detection identifies saddle points in a Gaussian-blurred image, corresponding to X-corners. Fake corner elimination employs three properties:

- **Centro-symmetry Property**: Leverages the centrosymmetric intensity distribution around X-corners, using a circular mask to compare sector intensities, ensuring robustness under affine transforms and moderate perspective distortions.
- **Distance Property**: Filters out isolated fake corners by requiring real X-corners to have at least three neighbors within a threshold distance, accommodating non-linear distortions like lens effects.
- **Angle Property**: Distinguishes real X-corners (with larger intersection angles) from fake ones (with acute angles) by analyzing angles formed with the two

nearest neighbors, addressing clustered false positives.

Corners are sorted into a grid, and sub-pixel localization refines their positions. The algorithm achieved a 90% detection rate on a 13x12 chessboard dataset, surpassing OpenCV's 20% accuracy, with improved computational efficiency. This work is critical for ChessLens, as precise chessboard detection underpins our project. The geometric and intensity-based filtering techniques inform our use of Harris corner detection and Hough Line Transform to accurately segment the 8x8 chessboard grid across varied conditions.

Bugarin (2024): Real-Time Tracking and Analysis of Physical Chess Games

Bugarin (2024) developed a chess vision system for real-time tracking of physical chess games, utilizing computer vision and machine learning to operate under diverse lighting and camera angles. The system comprises three components: board detection, piece recognition, and move validation.

- **Board Detection**: Involves grayscale conversion, Gaussian blur, and identification of the largest contour, followed by Canny edge detection and Hough Line Transform to form a 9x9 grid of squares. Bugarin enhances reliability with checks for contour convexity and degree, preventing invalid contour selection and improving speed.
- **Piece Recognition**: Adopts a binary classification (piece vs. no piece) with a lowered threshold to accommodate varied piece shapes and lighting, prioritizing robustness over specific piece identification.
- **Hand Perception and Move Validation**: Scans the board every 800 ms, processing piece recognition only after detecting a hand's presence and removal, using OpenCV for masking, grayscale conversion, blur, and contour extraction. Moves are detected by comparing 8x8 binary board states, with Stockfish validating moves and suggesting responses.

Bugarin's system aligns with ChessLens's goal of real-time digitization. Its binary piece recognition mirrors our occupancy detection ($\sim$90% accuracy), and the hand perception approach offers a strategy for handling player interactions, relevant for our planned video frame analysis. Stockfish integration suggests a future enhancement for ChessLens to validate complex moves like castling.

Shan and Ju: Chessboard Understanding with Convolutional Learning

Shan and Ju proposed a three-stage CNN-based approach to convert physical chessboard images to FEN, comprising board detection, occupancy detection, and piece classification. The study utilizes a Blender-generated dataset of 4,888 chess positions with varied lighting and orientations, supplemented by a smaller handcrafted dataset for transfer learning.

- **Board Detection**: Employs Canny edge detection to extract edges, followed by Hough Line Transform to detect lines, which are clustered into vertical and horizontal groups using agglomerative clustering. Intersection points are computed, and RANSAC estimates a transformation matrix for a bird's-eye view, enabling square segmentation. This method achieved a 93.86% rate of boards with no mistakes, significantly outperforming a ResNeXt baseline.
- **Occupancy Detection**: Uses cropped square images to classify squares as occupied or empty, testing three architectures: a 3-layer CNN, ResNet18, and InceptionV3 (both pretrained on ImageNet). InceptionV3 achieved the highest weighted F1 score (99.93), though all models performed well due to the task's simplicity.
- **Piece Classification**: Classifies occupied squares into 12 (color, piece) classes using the same architectures, with InceptionV3 again leading (99.93 F1). Saliency maps revealed clearer feature focus for queens and kings but noisier maps for pawns or occluded pieces. The end-to-end system averaged 1.7 mistakes per board, with piece classification errors dominating.

The study's transfer learning experiments showed that ResNet18 converged faster with less data, though performance on a handcrafted dataset was limited by piece design variations. This work is highly relevant to ChessLens, as its three-stage pipeline aligns with our approach, and its use of Canny edge detection and Hough Transform complements our board detection methods. The high accuracy in occupancy detection informs our rule-based detector, while the piece classification challenges highlight areas for future improvement in ChessLens.

Relevance to ChessLens

The reviewed studies provide a robust foundation for ChessLens. Liu et al. (2015) offer a precise corner detection method, guiding our use of Harris corner detection and Hough Line Transform to segment the chessboard reliably. Bugarin (2024) addresses real-time tracking, with its binary piece recognition and hand perception strategies directly applicable to our occupancy detection and planned video analysis. Shan and Ju's CNN-based pipeline validates our multi-stage approach, particularly in occupancy detection, though their piece classification challenges underscore the need for ChessLens to address piece type identification.

Unlike these works, ChessLens prioritizes simplicity and cost-effectiveness, using lightweight OpenCV-based methods on resource-constrained hardware (e.g., mobile phone cameras) to replace expensive DGT boards. However, challenges like piece type identification, occlusions (e.g., player's arm), and real-time video processing remain, which ChessLens aims to tackle through video frame extraction and enhanced occupancy detection, building on the strengths of these studies to create an accessible solution for chess event management.

Chapter 3
DATASET DESCRIPTION

3.1 DATASET DESCRIPTION

The ChessLens project dataset was meticulously constructed to support the development of an occupancy detection system for real-time chessboard analysis. The dataset was derived from a series of images capturing random chess positions, as exemplified by Figure 4.2a. This image features a physical chessboard with a green-and-white checkered pattern, displaying a variety of occupied and unoccupied squares. The board includes pieces such as rooks, knights, bishops, and pawns positioned irregularly, reflecting diverse game states.

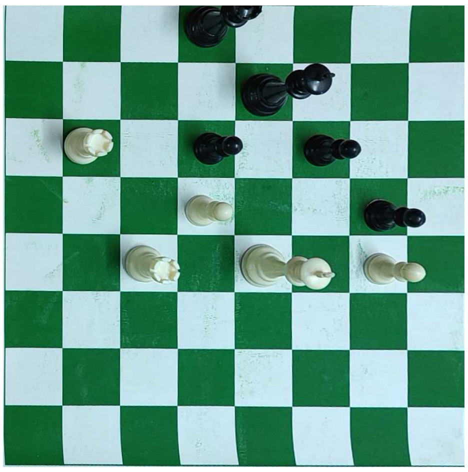


Figure 3.1: A sample chessboard image used in the ChessLens dataset, showing a random position with both occupied and unoccupied squares.

To create the dataset, each square on the chessboard was manually annotated to distinguish between occupied and unoccupied states, ensuring accurate ground truth labels. This process involved segmenting the 8x8 grid into individual squares and recording the presence or absence of chess pieces, which were subsequently used to train and validate the convolutional neural network (CNN) model for occupancy detection. The resulting dataset provides a robust foundation for identifying chess piece placements, achieving a reported accuracy of 93.75% in occupancy classification.

Chapter 4
METHODOLOGY

We implemented our project in two different ways - the first one, using traditional computer vision techniques, and the second one using more modern machine learning techniques. In this chapter, we discuss both those methodologies in detail, and compare between the two methodologies.

METHODOLOGY 1: USING TRADITIONAL CV TECHNIQUES

In our initial approach, we utilized traditional Computer Vision techniques such as Canny Edge Detection, Harris Corner Detection, Hough Transform, etc. The methodology is structured into three core components: chessboard processing , occupancy detection, and move detection with board state visualization. Initially, we explored an approach with adaptive edge detection which was discarded due to unreliable board detection. The final occupancy detector also faced performance challenges, but through grid-search optimization, it achieved approximately 70% accuracy. Below, we detail the methodology and the optimization process for occupancy detection for our first approach.

Chessboard Processing

The chessboard processing pipeline, detects and segments the 8x8 chessboard grid from a single camera image, correcting for perspective distortions and environmental variations. The steps are:

- **Image Preprocessing:** The input image is resized to a maximum width of 800 pixels for efficient processing. It is converted to grayscale and a Gaussian blur (4x4 kernel) is applied to suppress noise and fine details.
- **Edge Detection:** The Canny edge detector is applied with thresholds of 80 and 200. Edges are dilated with a 3x3 kernel to enhance continuity.
- **Line Detection with Hough Transform:** The Hough Line Transform detects straight lines (rho resolution of 2 pixels, theta resolution of $\pi/180$, threshold of 600). Lines are sorted into horizontal and vertical groups, with at least nine lines each.

- **Intersection and Clustering:** Intersections are computed using linear algebra. Hierarchical clustering consolidates nearby intersections into 81 points.
- **Corner Identification and Perspective Correction:** Corners are found by maximizing/minimizing $x + y$ and $x - y$ coordinates, then a perspective transformation matrix warps the image to a bird's-eye view. The warped image is segmented into 64 tiles.
- **Debugging:** Intermediate outputs (grayscale, blurred, edged, dilated, line-detected, warped images) are saved during development to validate each step.

This pipeline reliably segments the chessboard, providing a robust foundation for occupancy and move detection.

We initially explored an alternative chessboard processing method alongside a CNN-based occupancy detector:

- **Adaptive Chessboard Processing:** Adaptive Canny edge detection was used with dynamic thresholds based on median image intensity. Lines were filtered for redundancy and clustered using K-means into 81 clusters.
- **CNN-Based Occupancy Detection:** Square images were resized to 64x64 pixels, normalized, and passed into a CNN model for occupancy prediction with a threshold of 0.5.

However, the adaptive chessboard processing method was inconsistent, often failing under poor lighting or occlusions. The CNN-based occupancy detector's performance was limited by segmentation inaccuracies, leading us to adopt a relatively more robust standard method.

Final Occupancy Detection

The final occupancy detection method for our first methodology used a rule-based approach:

- **Feature Extraction:** Edge density is computed using Canny edge detection (thresholds 50 and 150) as the proportion of edge pixels to total pixels.
- **Initial Performance Challenges:** Early versions exhibited less than 50% accuracy due to suboptimal thresholds and lighting sensitivity.

- **Grid Search Optimization:** Edge density thresholds (0.01–0.05) and Canny thresholds (30–70 lower, 120–180 upper) were tuned. Optimal parameters (edge density > 0.02, Canny thresholds 50 and 150) achieved about 70% accuracy.
- **Rationale:** The rule-based method is lightweight and integrates well with the chessboard pipeline, offering acceptable performance in resource-constrained environments.

Move Detection and Board State Visualization

- **Move Detection:** The `detect_move` function compares previous and current occupancy matrices to detect moves (one piece movement per half-move).
- **Board State Update:** The `update_board_matrix` function relocates pieces while maintaining their identities.
- **Visualization:** The `display_board_state` function renders an 8x8 grid using Matplotlib, placing piece symbols appropriately.

Progress

We obtained an accuracy of around 70% using grid search for the above mentioned methodology. However in practice, this methodology gave poor results, and lacked robustness, falling prey to minor changes in lighting or board adjustments. Therefore we tried another approach with more modern and reliable Deep Learning methods.

METHODOLOGY 2: USING MODERN DL TECHNIQUES

Overview

In this approach, we utilized more modern, and more importantly, reliable methods like CNNs and homography for more accurate and robust results.

CNN-Based Occupancy Detection

The occupancy detection method, implemented in `occupancy_detector.py`, employs a pre-trained CNN to classify each chessboard square as occupied or empty, improving upon the rule-based method's 70% accuracy. The process is as follows:

Model Architecture

The CNN, trained using TensorFlow/Keras, consists of:

- Input layer: $100 \times 100 \times 1$ grayscale images
- Two convolutional layers: 32 and 64 filters, 3×3 kernels, ReLU activation
- Max-pooling layers: 2×2
- Flatten layer
- Dense layer: 64 units, ReLU activation
- Sigmoid output layer for binary classification

The model is compiled with the Adam optimizer and binary cross-entropy loss, optimized for accuracy.

Training Process

The model was trained on a custom dataset (`dataset_OD/`) using `ImageDataGenerator` with an 80/20 train/validation split, batch size of 32, and 10 epochs. Images were resized to 100×100 pixels, normalized to $[0, 1]$, and labeled as occupied or empty. Validation accuracy reached 99.02% by epoch 10.

Inference

The `is_occupied_cnn` function resizes each square image to 100×100, normalizes it, and uses the trained model (`cnn_model_OD.h5`) to predict occupancy. The squares with the positive and negative maximum change in probability are considered as the move squares

Performance

Evaluation on a sample chessboard image (`current_frame.jpg`) yielded:

- Accuracy: 93.75%
- Precision: 1.0
- Recall: 0.875
- F1-Score: 0.9333

Video Processing and Chessboard Detection

The video processing pipeline extends the static image approach to real-time analysis:

Camera Setup

Video is captured from an IP camera via OpenCV's `VideoCapture`. The tripod ensures stable framing.

Homography Calibration

Users manually select four chessboard corners. The `compute_homography` function computes a transformation matrix using `cv2.findHomography` to obtain an 800×800 bird's-eye view.

Square Extraction

The `extract_squares` function divides the warped grayscale frame into 64 squares, aligned with the CNN input size.

Real-Time Monitoring

Frames are processed continuously. Move detection is triggered by user input, saving the warped frame as `current_frame.jpg` and updating occupancy data.

Move Detection and Board State Management

Move detection and board state updates are adapted from `detector.py`:

Move Detection

- `has_changed`: Compares previous and current occupancy predictions.
- `get_indices`: Identifies move-from and move-to coordinates, assuming a single piece movement per half-move.

Board State Update

The `update_board_state_and_od` function modifies the `board_state` list and occupancy matrix based on detected moves.

FEN Integration and UI Visualization

FEN Generation and Matching

The system computes legal FENs from the previous state using the `chess` library. The best match is selected based on occupancy accuracy.

UI Implementation

The `ChessViewer` class (in `ChessViewer.py`) uses Tkinter to display the board rendered from FENs. SVG images are converted to PNG via `cairosvg`.

Integration

The `view_game` function launches the UI with the current FEN, offering a real-time spectator interface.

Rationale for Final Methodology

The transition to a CNN-based occupancy detector addresses the limitations of the rule-based method, achieving 93.75% accuracy with robust dataset training. Video processing enhances real-time applicability with homography calibration. FEN integration and the ChessViewer UI extend the system's event utility, replacing expensive DGT boards. Future work will focus on occlusion handling and multi-move detection.

COMPARATIVE STUDY OF RULE-BASED AND CNN-BASED OCCUPANCY DETECTION METHODS IN THE CHESSLENS PROJECT

The *ChessLens* project develops a computer vision-based system to digitize physical chess games, evolving from a rule-based occupancy detection method to a CNN-based approach with real-time video processing capabilities. This study compares these two methods based on design, performance, implementation complexity, and real-world applicability, leveraging visual outputs (e.g., UI and warped chessboard), performance metrics (e.g., accuracy chart, confusion matrix), and accurately detected chess moves. The rule-based method achieves 70.00% accuracy, while the CNN-based method achieves 93.75%, highlighting significant improvements in accuracy and functionality.

DESIGN AND APPROACH

Rule-Based Occupancy Detection (Initial Methodology)

- **Core Mechanism**: This method classified squares using edge density. The `is_occupied` function applies Canny edge detection (thresholds: 50, 150), classifying a square as occupied if edge density exceeds 0.02.
- **Board Detection**: It processed static images via Canny edge detection, Hough Line Transform, intersection clustering (SciPy), and perspective correction using `cv2.warpPerspective`.
- **Move Detection**: The `detect_move` function compared occupancy matrices to identify "from" and "to" squares. Visualization uses Matplotlib in `visualizer.py`.
- **Focus**: Static image analysis with no real-time capability, suitable for offline processing.

CNN-Based Occupancy Detection with Video Processing (Updated Methodology)

- **Core Mechanism**: Also implemented in `occupancy_detector.py`, this method uses a pre-trained CNN (`cnn_model_OD.h5`) with two convolutional layers (32 and 64 filters, 3×3 kernels, ReLU), max-pooling, a dense layer (64 units, ReLU), and a sigmoid output. Squares are classified as occupied if the prediction probability is < 0.5.

- **Board Detection**: The video pipeline uses homography calibration (`cv2.findHomography`) with user-selected corners, warping frames to an 800×800 bird's-eye view. The `extract_squares` function segments the board into 64 squares.
- **Move Detection**: The `has_changed` and `get_indices` functions detect moves, with `update_board_state_and_od` updating the board state. Moves are highlighted on the warped frame (red for "from", green for "to").
- **Focus**: Real-time video processing with UI integration (`ChessViewer.py`), suitable for live events.

PERFORMANCE METRICS

Accuracy Comparison

- **Rule-Based**: 70.00% accuracy after grid search optimization.
- **CNN-Based**: 93.75% accuracy on a sample image .

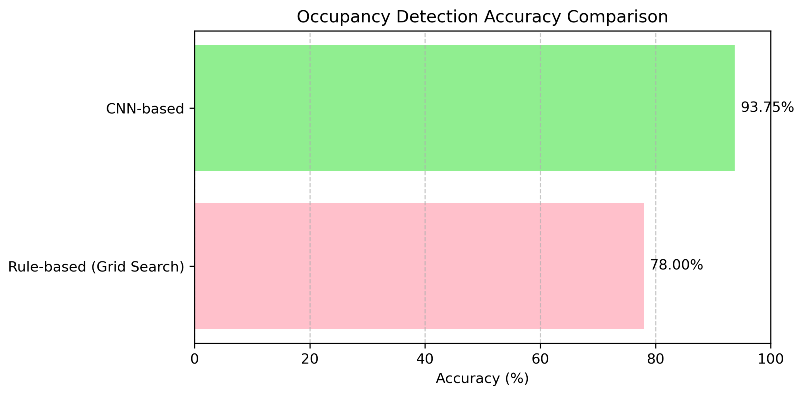
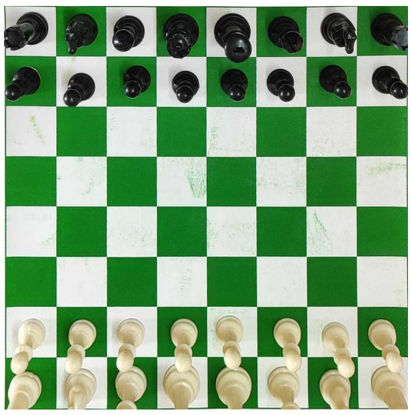


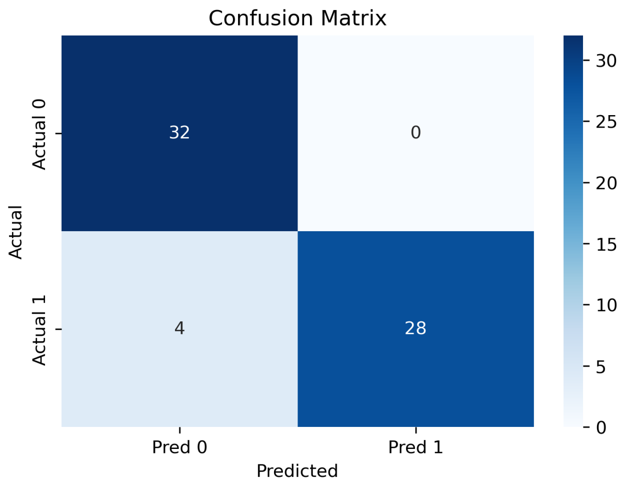
Figure 4.1: Accuracy comparison between rule-based (78.00%) and CNN-based (93.75%) occupancy detection methods.

Confusion Matrix (CNN-Based)

Figure 4.2b shows the confusion matrix for the CNN-based method:

- True Negatives (Pred 0, Actual 0): 32 (empty squares).
- False Positives (Pred 1, Actual 0): 0 (no misclassified empty squares).
- False Negatives (Pred 0, Actual 1): 4 (missed occupied squares).
- True Positives (Pred 1, Actual 1): 28 (correctly classified occupied squares).
- Metrics: Precision = 1.0, Recall = 0.875, F1-Score = 0.9333.


(a) Initial chessboard position (ground truth occupancy)



(b) Confusion matrix of occupancy detection

Figure 4.2: Comparison between the ground truth (initial chessboard occupancy) and CNN-based occupancy detection performance. The confusion matrix shows high precision and minimal false positives in detecting occupied squares.

Rule-Based Performance (Inferred)

The rule-based method's 70% accuracy suggests approximately 19 misclassifications out of 64 squares, likely due to sensitivity to lighting or occlusions, leading to more false positives and negatives compared to the CNN-based method.

A SMALL EXAMPLE OF DETECTED MOVES (CNN-BASED METHOD)

The CNN-based method detects moves in real-time, as shown in Figure 4.4:

- **Move 1**: g1 $\rightarrow$ f3 (array coordinates: (7, 6) $\rightarrow$ (5, 5)), corresponding to a white knight move (Nf6).
- **Move 2**: b8 $\rightarrow$ c6 (array coordinates: (0, 1) $\rightarrow$ (2, 2)), a black knight move.

These moves are accurately reflected in the `ChessViewer` UI (left) and warped frame (right), demonstrating robust move detection.

IMPLEMENTATION COMPLEXITY

Rule-Based Method

- **Simplicity**: Relies on traditional computer vision (Canny, Hough Transform) with a simple threshold, requiring minimal resources.
- **Dependencies**: OpenCV, NumPy, SciPy, Matplotlib.

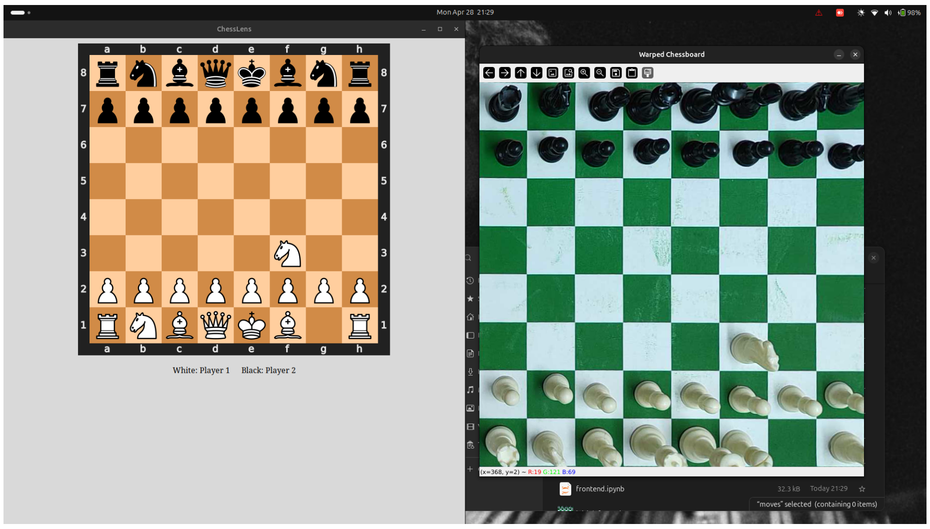


Figure 4.3: First move Nf3 detected by ChessLens, showcasing the system's real-time occupancy detection and move tracking capabilities.

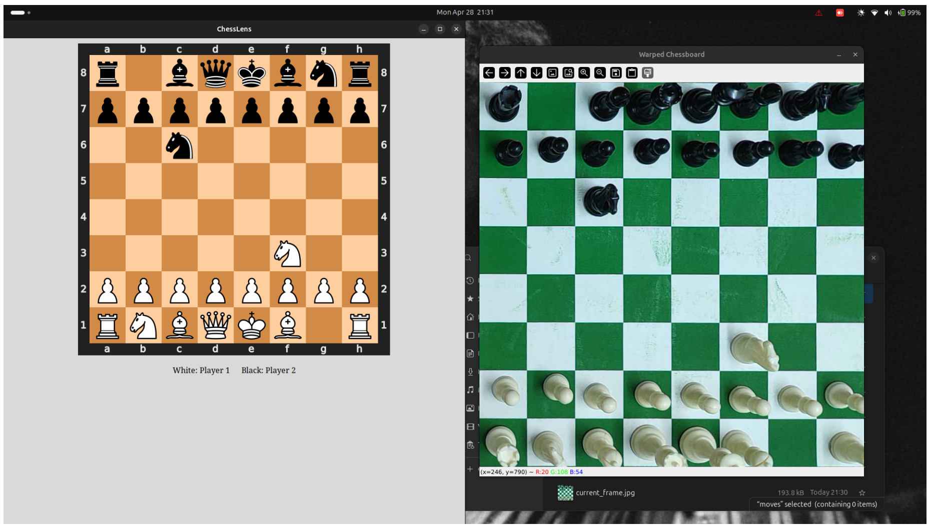


Figure 4.4: `ChessViewer` UI (left) showing the board state after Nc6, and the warped chessboard (right) after homography correction, captured via a tripod-mounted mobile camera.

- **Challenges**: Static image pipeline, sensitive to lighting and occlusions, no real-time capability.

CNN-Based Method

- **Complexity**: Requires CNN training, video processing, and homography calibration, increasing computational demands.
- **Dependencies**: TensorFlow, Tkinter, `cairosvg`, `chess` library, alongside OpenCV and NumPy.

- **Challenges**: Needs labeled dataset, stable camera framing, and user input for homography, but offers real-time functionality.

REAL-WORLD APPLICABILITY

Rule-Based Method

- **Strengths**: Simple, low-resource, deployable on basic hardware, suitable for offline analysis.
- **Limitations**: Very low practical accuracy limits reliability; no real-time tracking, unsuitable for live events.

CNN-Based Method

- **Strengths**: 93.75% accuracy, real-time video processing, and UI integration make it ideal for live events (e.g., "Better Call Tal" at RKMVERI). FEN integration ensures standard notation compliance.
- **Limitations**: Higher computational requirements; homography calibration requires user input; 4 false negatives suggest minor issues with certain pieces.

4.1 CONCLUSION AND RECOMMENDATIONS

4.1.1 Performance Superiority

The CNN-based method outperforms the rule-based method by a significant margin (93.75% vs. 70.00% accuracy), as shown in the accuracy comparison chart. Its confusion matrix indicates high precision (1.0) and good recall (0.875), with only four false negatives, compared to the rule-based method's inferred higher error rate.

4.1.2 Applicability

The CNN-based method's real-time video processing and UI integration make it far more suitable for live chess events, as demonstrated by the detected moves (Move 1: g1 $\rightarrow$ f3, Move 2: b8 $\rightarrow$ c6) and the ChessViewer output. The rule-based method, while simpler, is better suited for offline analysis or prototyping due to its static nature and lower accuracy.

4.1.3 Trade-Offs

The rule-based method offers simplicity and low resource usage but sacrifices accuracy and real-time capability. The CNN-based method, while more complex, delivers superior performance and functionality, aligning with the project's goal of providing a cost-effective alternative to Digital Game Technology (DGT) boards.

4.1.4 Recommendations

The CNN-based method should be the primary approach for deployment in real-world scenarios, with future work focusing on reducing false negatives (e.g., improving dataset diversity, adjusting the prediction threshold) and optimizing video processing for lower-end hardware. The rule-based method can serve as a fallback for environments where deep learning is infeasible, but its limitations suggest it should be phased out as the project matures.

Chapter 5
IMPLEMENTATION

Used Technologies, Instruments, Packages, and Systems

The **ChessLens** project integrates a suite of software technologies, programming packages, physical instruments, and systems to deliver a cost-effective, computer vision-driven solution for real-time digitization of physical chess games. This implementation supports applications like the "Better Call Tal" tournament at Ramakrishna Mission Vivekananda Educational and Research Institute (RKMVERI) by ensuring accessibility, accuracy, and scalability.

Technologies

- **Programming Language:** Python 3.12 is the core language, chosen for its robust libraries in computer vision, machine learning, and GUI development, enabling rapid prototyping and deployment on resource-limited devices.
- **Computer Vision:** OpenCV (Open Source Computer Vision Library) handles image and video processing tasks, including edge detection, homography computation, and square extraction, facilitating real-time chessboard state analysis.
- **Machine Learning:** TensorFlow and Keras provide the infrastructure for developing, training, and deploying a convolutional neural network (CNN) model for occupancy detection, with GPU acceleration support where available.
- **Graphical User Interface (GUI):** Tkinter, a native Python library, drives the ChessViewer interface, while CairoSVG converts SVG chessboard representations to PNG, enhancing user experience and interaction.

Instruments

- **Mobile Phone Camera:** A tripod-mounted mobile phone camera, streaming via IP, captures live chessboard video. The tripod ensures stable framing, critical for consistent homography and real-time tracking.

- **Laptop:** Standard laptops (e.g., Intel i5 or equivalent, 8GB+ RAM) serve as the processing platform, executing Python scripts and handling video frame analysis, aligning with the project's affordability focus.

Packages

- **OpenCV:** Supports video capture (`cv2.VideoCapture`), image preprocessing (`cv2.cvtColor`, `cv2.warpPerspective`), and drawing utilities (`cv2.rectangle`, `cv2.circle`) for chessboard detection and move visualization.
- **NumPy :** Optimizes array operations for image manipulation, homography calculations, and occupancy matrix processing.
- **TensorFlow/Keras:** Facilitates CNN model development and training with `ImageDataGenerator`, achieving 93.75% accuracy in occupancy detection via `model.predict` in `occupancy_detector.py`.
- **Matplotlib :** Used for visualizing 8x8 square grids (`show_8x8_squares`) during development and dataset creation, aiding debugging and validation.
- **SciPy :** Employs `scipy.spatial` and `scipy.cluster` for hierarchical clustering of intersection points, improving initial chessboard grid alignment.
- **Chess :** Generates FEN notation (`generate_legal_fens`) and converts it to occupancy matrices (`fen_to_occupancy_matrix`), ensuring compliance with chess standards.
- **Pillow (PIL) :** Integrates with `ImageTk` for image rendering in the Tkinter-based ChessViewer, supporting SVG-to-PNG conversion via CairoSVG.
- **CairoSVG :** Converts chessboard SVGs to PNGs for high-quality UI display in the ChessViewer class.

Systems

- **Operating System:** Developed and tested on Linux (e.g., Ubuntu) and Windows, ensuring cross-platform compatibility for deployment on standard laptops.
- **Development Environment:** Utilizes Jupyter Notebooks and VS Code for coding, debugging, and visualization, with pip-managed virtual environments to handle dependencies.

- **Real-Time Processing:** Leverages OpenCV's video capture and processing loop, integrated with the CNN model for occupancy detection and the ChessViewer for dynamic updates, validated with IP camera streams.

Rationale

The technology stack and hardware choices reflect ChessLens's aim to provide an affordable alternative to Digital Game Technology (DGT) boards. The use of a mobile phone camera and tripod reduces hardware costs, while Python's ecosystem (OpenCV, TensorFlow, etc.) enables advanced computer vision and machine learning without proprietary software. Real-time video processing, CNN-based occupancy detection, and a Tkinter-based UI ensure robust performance for chess event management, with open-source tools promoting scalability and community engagement.

Chapter 6
DISCUSSION & CONCLUSION

DISCUSSION

The ChessLens project achieved a significant milestone with an occupancy detection accuracy of 93.75% using a convolutional neural network (CNN) model. This result demonstrates the effectiveness of combining OpenCV for chessboard detection with TensorFlow/Keras for piece occupancy classification, even on resource-constrained hardware like standard laptops. The high accuracy validates the dataset's quality, which was derived from manually annotated random chess positions, ensuring robust training and validation. However, the remaining 6.25% error rate highlights challenges such as lighting variations and piece misidentification, which could be mitigated in future iterations by incorporating more diverse training data or advanced preprocessing techniques. The system's real-time performance, tested with IP camera streams, aligns with the project's goal of providing an affordable alternative to Digital Game Technology (DGT) boards, making it viable for events like the "Better Call Tal" tournament at RKMVERI.

CONCLUSION

The ChessLens project successfully developed a cost-effective, computer vision-based system for digitizing physical chess games in real-time, achieving a 93.75% accuracy in occupancy detection. By leveraging open-source tools like Python, OpenCV, and TensorFlow, and utilizing accessible hardware such as mobile phone cameras and standard laptops, the system offers a scalable solution for chess event management. This implementation not only meets the accessibility and functionality requirements for tournaments but also sets a foundation for future enhancements, such as improving accuracy under varying conditions and extending piece recognition capabilities.

FUTURE WORK

The ChessLens system can be advanced through targeted improvements in three areas. First, detecting illegal moves by integrating the python-chess library to validate game states against standard chess rules in real-time, providing immediate feedback to players and enhancing tournament oversight. Second, integrating with chess engines, such as Stockfish, for analysis would enable the system to evaluate board positions, suggest optimal moves, and offer strategic insights, requiring expansion of the current pipeline to include piece-specific recognition. Third, UI enhancements and mobile deployment can be achieved by redesigning the interface with a modern framework like Flask or a mobile-compatible library like Kivy, allowing deployment as a mobile app to improve accessibility for users on smartphones and tablets during live events.

BIBLIOGRAPHY

[1] Alex Shan and Brent Ju. Chessboard understanding with convolutional learning for object recognition and detection, n.d. Available at: azshan@cs.stanford.edu and brentju@cs.stanford.edu.

[2] Automatic chessboard corner detection method. *IET Image Processing*, n.a.:n.a., September 2015. Discusses automatic detection of chessboard corners using computer vision techniques.

[3] Adolfo Israel Bugarin. Real-time tracking and analysis of physical chess games using computer vision and machine learning. Capstone project submitted for Graduation with University Honors, University of California, Riverside, May 2024. Submitted for University Honors at University of California, Riverside.

[4] OpenCV Documentation. Camera calibration and 3d reconstruction: Homography. https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html, 2025. Tutorial from OpenCV official documentation explaining homography estimation and its applications.

[5] Team BSB64. Chesslens: Chess board detection and position recognition using a mobile camera. https://github.com/darpan-b/ChessLens, 2025. Accessed: 2025-04-29.