CVChess: A Deep Learning Framework for Converting Chessboard Images to Forsyth–Edwards Notation

Luthira Abeykoon
Student# 1005678901
luthira.abeykoon@mail.utoronto.ca

Darshan Kasundra
Student# 1010292434
darshan.kasundra@mail.utoronto.ca

Gawtham Senthilvelan
Student# 1009977704
gawthaman.senthilvelan@mail.utoronto.ca

Ved Patel
Student# 1010140447
ved.patel@mail.utoronto.ca

—-Total Pages: 9

1 Project Description

Chess has seen a dramatic resurgence in popularity, with tournament viewership rising by 55% from 2020 to 2024 (ChessWatch, 2024). A key factor behind this surge can be attributed to the growing presence of chess in live streaming, which has made learning the game more accessible than ever. People are interested in chess, and more importantly, people want to get better at chess. While online platforms provide learning methods such as the 'hint' button on Chess.com, which recommends the optimal move at a given game state, no such convenience exists for physical chess games.

Our team proposes an easy-to-use tool which can quickly inform users of the most optimal move to make. By using a CNN to detect piece positions from images of physical chessboards at random game states, we can generate a Forsyth-Edwards Notation (FEN) string, a standardized representation of a chessboard. This string can easily be fed into a chess engine to compute the most optimal move. Our system acts as a real-world 'hint' button, providing guidance for players of all skill levels.

2 Individual Contributions and Responsibilities

Following the project proposal, our team has followed the initial plan of conducting weekly meetings on our chosen communication platform, Discord, where we keep notes of meeting discussions and plan next steps for the following meeting. Meeting frequencies have increased as the deadline for the project progress report has approached.

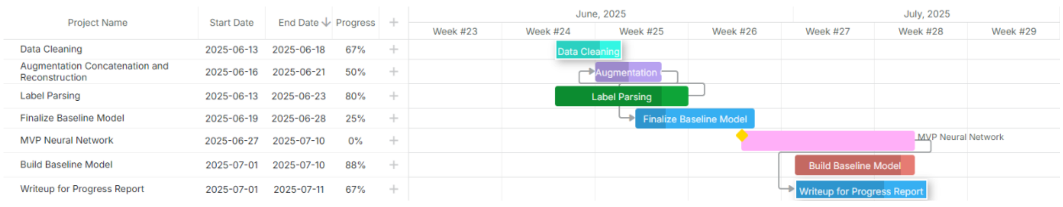
2.1 Project Management Software

Urgent matters have been discussed through text channels for individual tasks. As the need for more storage became more pressing, we decided to create a new Google Drive account with 200 GB of storage that every member has access to, so we can use a larger dataset.

Using GitHub for code collaboration, our team has not had any issues with merge conflicts. Decisions have been made through group consensus. Though this process may not work for all groups, we found that decision through consensus has allowed our team to move forward quickly as we share similar opinions on project ideas.

2.2 Team Contributions and Project Progress

Figure 1 below depicts a Gantt Chart generated by Mach AI, which outlines the project timeline and progress for various tasks throughout the duration thus far in the project. The chart displays a few key project components, including Data Cleaning, Augmentation and Concatenation, Label Parsing, Finalizing Baseline Model, Creating a Neural Network MVP for the model, Building the Baseline Model and Writing the Progress Report. The Gantt chart also shows how certain tasks are related to each other, as shown by the connecting arrows. We followed this Gantt Chart to ensure that we were on track to finish the deliverable.



Project Name	Start Date	End Date	Progress
Data Cleaning	2025-06-13	2025-06-18	67%
Augmentation Concatenation and Reconstruction	2025-06-16	2025-06-21	50%
Label Parsing	2025-06-13	2025-06-23	80%
Finalize Baseline Model	2025-06-19	2025-06-28	25%
MVP Neural Network	2025-06-27	2025-07-10	0%
Build Baseline Model	2025-07-01	2025-07-10	88%
Writeup for Progress Report	2025-07-01	2025-07-11	67%

Figure 1: Mach AI Generated Gantt Chart

Table 1 below outlines the team's progress leading up to the completion of the Progress Report.

Table 1: Milestone 2 - Progress Report

Member	**Task**	**Deadline**	**Status**
Luthira Abeykoon	Data Cleaning + Preprocessing Pipeline	Jun. 18th	Completed
Darshan Kasundra	Augmentation Concatenation and Reconstruction	Jun. 21st	Completed
Ved Patel	Label Parsing	Jun. 23rd	Completed
Gawtham Senthilvelan	Finalize Baseline Model	Jul. 1st	Completed
Ved Patel	Minimum Viable Product Neural Network	Jul. 4th	Completed
Everyone	Progress Report	Jul. 11th	Completed

Table 2 below outlines the updated project plan and distribution of work to complete the Final Deliverable.

Table 2: Milestone 3 - Final Deliverable

Member	**Task**	**Deadline**	**Status**
Luthira Abeykoon	Improve preprocessing of images	Jul. 18th	Completed
Ved Patel	Optimizing Model to Identify Empty Pieces	Jul. 25th	Completed
Gawtham Senthilvelan	Bayesian Optimization to optimize Neural Network Hyperparameters	Aug. 1st	Completed
Darshan Kasundra	Collect New Testing Data and Test Model	Aug. 8th	Completed
Everyone	Final Deliverable, Project Presentation and Video	Aug. 13th	Completed

2.3 Contingency Plans

The team has come up with contingency plans in case certain tasks are abandoned by a member as the deadline for the final report approaches. We have agreed that though certain members are assigned to certain tasks, it is expected that each member of the team should be able to step in and contribute to the completion of the task. The person assigned the task is simply held responsible to ensure that the task is completed by the deadline.

3 Data Processing

The primary data source used for the project is the Chess Recognition Dataset (ChessReD2K). This dataset consists of 2,078 images taken from 3 different smartphone cameras (Apple iPhone 12 - 3,024x3,024, Huawei P40 pro - 3,072x3,072, Samsung Galaxy S8 - 3,024x3,024) of 20 real-world chess games each at a different board state. The dataset was split in a 70/15/15 ratio, resulting in 1,442 training images, 330 validation images, and 306 test images.

Each image contains a real chessboard with pieces arranged according to the progression of an actual game. These images were taken from various angles and under different lighting conditions to mimic a bystander taking a picture of a live chess game. Each image is annotated with ground truth chess piece positions derived from FEN strings.

Since the goal is to classify the content of the individual chessboard squares, below is a detailed outline of the steps taken to extract square-level image crops from full-board images.

3.1 Board Corner Detection and Perspective Transform

Each image is first converted to grayscale using cv2.cvtColor(), then smoothed with a Gaussian blur (kernel size 5x5) to reduce noise and enhance edge continuity. Canny edge detection is applied with thresholds of 50x150, which clearly identifies the edges of the chessboard. This is followed by morphological dilation using the same 5x5 kernel to close the gaps in the edge segments due to imperfections such as shadows, chess piece overlaying edges, etc.

Contours are extracted using cv2.findContours() and sorted by area. Only the largest contour with 4 corners and an area exceeding 5% of the total image is retained as the candidate chessboard. The four corner points are arranged in a consistent orientation, with the white square positioned at the top-left (a8) and bottom-right (h1) corners. This ordered set is then used to compute a projective transformation using cv2.getPerspectiveTransform().

The output of this is a warped 400x400-pixel image representing a top-down-aligned view of the chessboard. This standardization step ensures consistent geometry for downstream processing and model inputs.

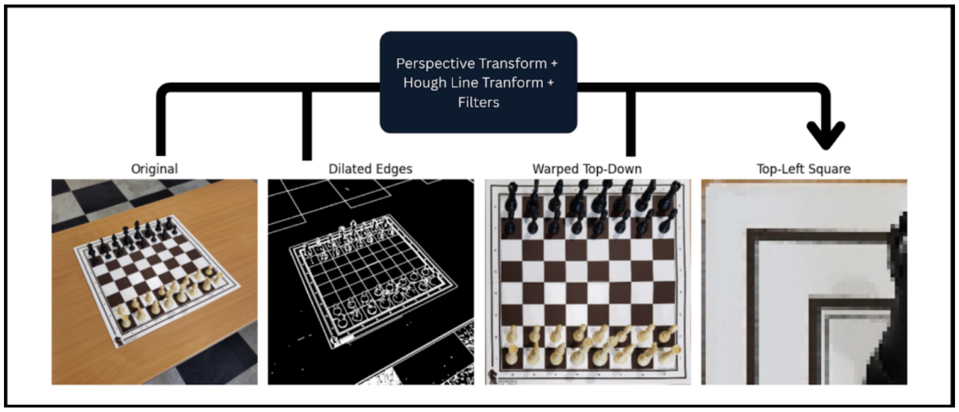


Figure 2: Visual Representation of Chess Board Recognition and Perspective Transform

3.2 Square Slicing and Feature Extraction

Once the warped chessboard image is geometrically aligned, it is divided into 64 square patches based on the 8x8 grid layout. This slicing ensures that each resulting image patch corresponds precisely to one square on the board. The ground truth FEN string representing the full board state is parsed into a 64-element label array by expanding any digits to the corresponding number of empty squares and mapping each character to one of 13 distinct classes: six for white pieces (P, N, B, R, Q, K), six for black pieces (p, n, b, r, q, k), and one for an empty square. These class labels are then encoded based on the model type: either as one-hot vectors for neural networks or integer class indices for traditional classifiers like SVMs.

Table 3: FEN Character to Piece Classification

FEN Character	**Piece**	**Class ID**
1	Empty Square	0
p	Black Pawn	1
r	Black Rook	2
n	Black Knight	3
b	Black Bishop	4
q	Black Queen	5
k	Black King	6
P	White Pawn	7
R	White Rook	8
N	White Knight	9
B	White Bishop	10
Q	White Queen	11
K	White King	12

3.3 Preprocessing Validation and Dataset Integrity

To ensure data reliability, the output of each preprocess_chessboard() call is subjected to two quality checks, which include: the images failing to return four valid corners are logged into a text file and then skipped, and warped boards with low visual variance (standard deviation of less than 50) are flagged as invalid.

Out of 2,078 images, around 50% of the images were valid, 34% were incorrectly warped, and 16% returned errors due to incorrect edge detection. After extracting square-level patches from the valid images, this yields approximately 132,992 high-quality samples.

However, this raises concerns about bias in the remaining dataset. The retained samples are more likely to be well-lit, clean, and uniformly positioned boards, which may not generalize well to real-world conditions.

3.4 Gathering New Data

Given the model's real-world applicability and its flexibility to handle images of varying resolutions, it would be ideal to use actual photographs taken by each team member's phone to introduce diversity into the testing dataset. These photos will consist of physical chessboards with pieces arranged on them and replicate the conditions present in the ChessReD2K dataset - captured from various angles and under different lighting conditions - to truly evaluate the model's performance in realistic scenarios. Afterwards, these images would be pre-processed and passed through the model to test its accuracy.

3.5 Challenges Faced

The primary challenge encountered during pre-processing was the significant loss of data caused by the failure to accurately detect the chessboard or apply a successful perspective transformation.

Out of the entire training dataset, only 50% of these images were valid, meaning that preprocessing removed half of the data before training even began. The main underlying issue is that each image contains background noise of table edges and checkered floor tiles which decreases the Canny edge detection's performance. Due to the method utilized in pre-processing - finding the largest contour with 4 corners and area exceeding 5% of the total image - it may mistakenly detect table edges as the largest contour, causing the image to be invalid.

4 BASELINE MODEL

To establish a foundation for evaluating our deep learning model, we implemented a baseline using a traditional machine learning algorithm. Specifically, we use a Support Vector Machine (SVM) classifier with a radial basis function (RBF) kernel, trained on a histogram of oriented gradients (HOG) features extracted from chessboard squares.

4.1 HANDLING CLASS IMBALANCE

Due to an imbalance in the dataset from just 485 samples for some pieces to over 37,000 samples for empty squares, we performed random oversampling using sci-kit learns's sklearn.utils.resample. Each class was resampled to match the median class size of 938 samples, resulting in a balanced training set.

4.2 TRAINING AND TESTING

To parse the data into the SVM, we constructed a pipeline consisting of a StandardScaler, PCA for dimensionality reduction, and finally the SVM itself. We split the balanced data into training and testing with a 70/30 split to maintain class distribution across sets. We then used GridSearchCV with a 5-fold StratifiedKFold cross-validation to search over and find the most optimal hyperparameters. Some parameter combinations failed due to PCA components exceeding the number of features; however, it was able to complete with the following configuration:

```
{'pca__n_components': 100, 'svm__C': 10, 'svm__gamma': 0.01}
```

Which was used in the final SVM model. In order to evaluate the performance of our model, we compared several benchmarks such as tile-level accuracy, macro-averaged F1 score, and the fully assembled FEN string average. The final SVM achieved 68% accuracy on the test set for tile level accuracy on the test set. Performance varied across some of the classes:

- Well-defined pieces like Queens and bishops had higher precision
- Ambiguous squares that are not immediately apparent were more prone to error for pieces like pawns or empty squares.

This can be visually seen in Figure 3 shown on the next page.

The matrix is normalized so each row adds up to 100%. The diagonal shows accuracy per class bishops and queens score around 66-71% while empty squares only reach 41% and black kings just 39%. Empty squares are often called white rooks (23%) or kings (8%), and the model tends to over-predict rooks and white pawns. Overall, the model performs well on clearly defined piece shapes but struggles most with classes that are small or exhibit low contrast.

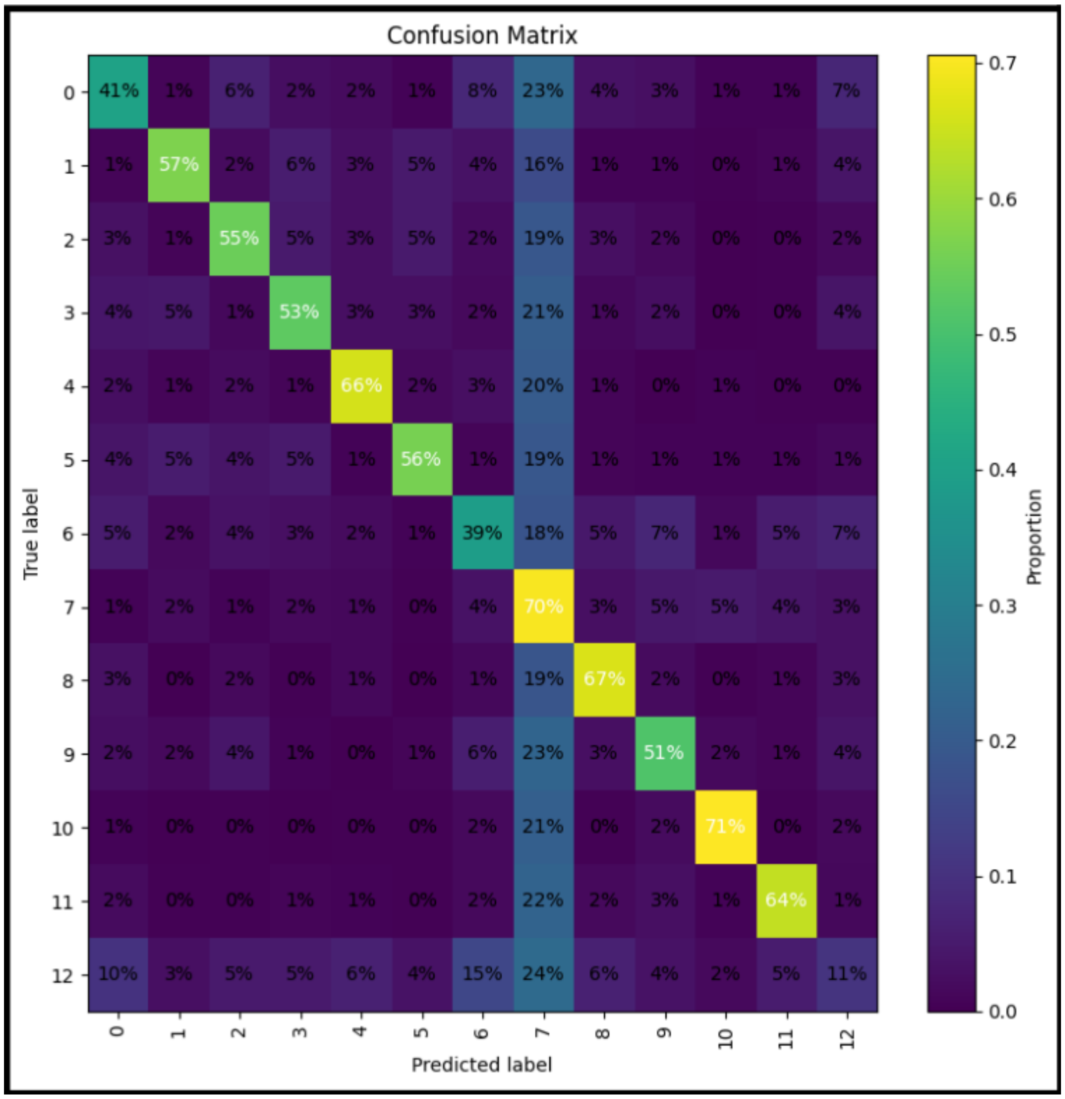


Figure 3: Confusion Matrix for Each Piece

4.3 Full-Board Prediction and Limitations

Reconstructing the board means predicting all 64 squares and stitching those labels into a FEN string. It only takes one misclassified tile to break the entire string, so errors compound quickly. Even with 68% accuracy per square, our end-to-end FEN accuracy is only about 10%, roughly one perfect board out of ten. That gap shows why treating each square in isolation isn't enough. We need models that understand how pieces relate across the board to boost full-board performance, which could be changed and iterated upon in the future. However, considering the current state of the art of 15% accuracy (Ding et al., 2016), the baseline performs about 2/3 of that level, which demonstrates that this method still provides a solid starting point.

5 Primary Model

The team decided that a convolutional neural network would be most appropriate for classifying the contents of each square on the top-down warped chessboard images. The model takes in 3x256x256 chessboard images and outputs 64 predictions (1 per square), each with a probability distribution over the 13 classes.

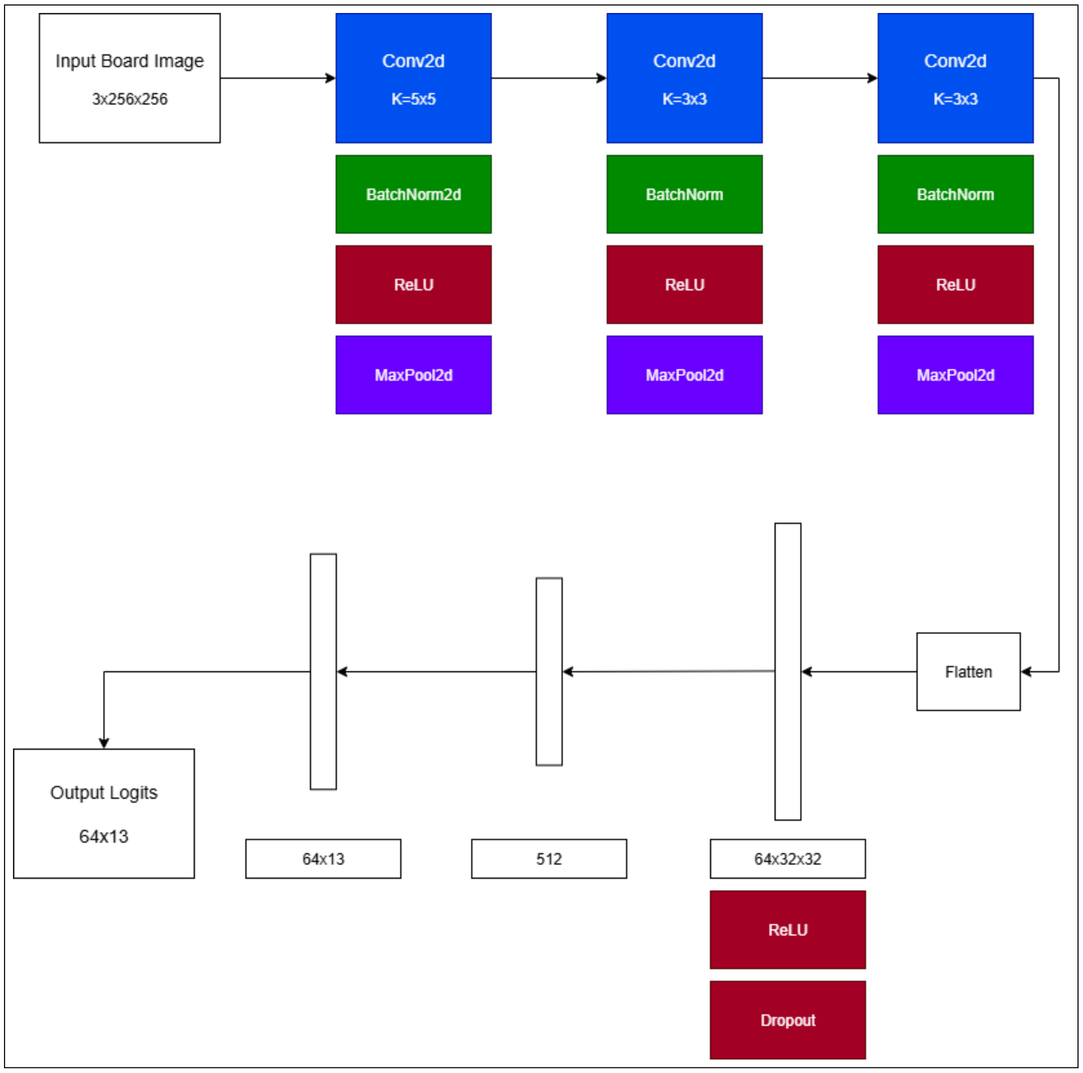


Figure 4: Primary Model Overview

5.1 Design Reasoning

By using concepts taught throughout this course, we believe that this model has the characteristics that make up a strong choice for this problem. Across the use of 3 convolutional layers, 3 fully connected linear layers, batch normalization, maxpooling, dropout, and ReLU activation function, our model is complex enough and built for this problem, while keeping in mind the time constraints and GPU limits on Google Colab.

The convolutional layers are essential for learning spatial correlations within the image, which is particularly important when working with structured visual data like chessboards. Max pooling is used to progressively reduce the spatial dimensions of the feature maps. This not only makes computation more efficient but also helps the network focus on high-level, abstract features that are more relevant for accurate classification.

To further improve training efficiency and stability, we incorporate batch normalization. By normalizing the inputs of each layer, we ensure that larger training steps can be taken (due to one global learning rate), which can significantly reduce the number of iterations required for convergence. In addition, we use dropout as a regularization technique to help prevent overfitting. Dropout forces the model to not rely too heavily on any single feature which in turn promotes better generalization by encouraging the network to learn more robust and distributed representations.

Finally, we choose the ReLU activation function because of its simplicity and effectiveness. ReLU introduces non-linearity into the model, which is crucial for learning complex patterns. Moreover, it helps mitigate the vanishing gradient problem, making it a reliable choice for deep neural networks.

5.2 Training and Testing

The model was trained using the same 70/30 split as the baseline model for 25 epochs using the Adam optimizer, cross entropy loss function, batch size of 32, and a learning rate of 0.001 (these hyperparameters were decided on via random search). We kept track of 2 main metrics, tile accuracy for all squares, which measures how many of the 64 tiles were predicted correctly, and tile accuracy for non-empty squares, which focused on predictions on squares which had chess pieces on them only.

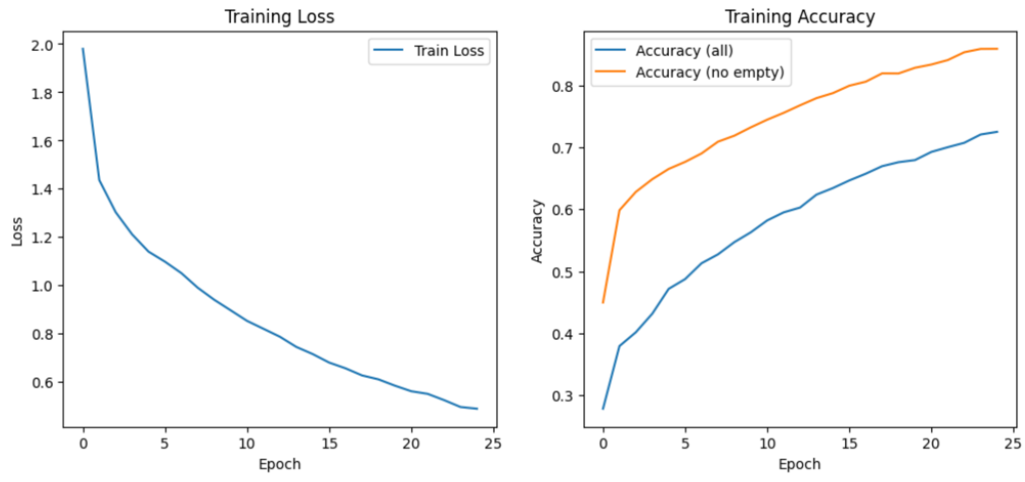


Figure 5: Training Loss and Training Accuracy

Through training, the model reached 72.5% accuracy for all tiles and 85.9% for non-empty squares. For validation data, the model achieved 59.5% and 50.5% respectively, a step down from the baseline model. Despite scoring slightly lower than the baseline SVM in validation tile accuracy, the deep learning model has demonstrated better generalization to rare and complex piece shapes, especially in cluttered positions. Additionally, our model can be trained end-to-end directly from pixel data without requiring handcrafted features such as HOG. This allows it to learn spatial hierarchies and patterns across the full board context.

We also attempted full-board FEN string reconstruction from the model's 64 square predictions. The board-level accuracy improves modestly compared to the baseline, due to better per-tile precision on rare piece types. This supports the feasibility of our CNN-based approach for chessboard digitization, while also exposing limitations that future architectural improvements (attention mechanisms or relational modeling) could address.

5.3 Challenges

One of the main challenges during development was managing model complexity. Our original architecture had over 135 million parameters, which led to rapid overfitting and GPU memory errors on Google Colab. We redesigned the model to include fewer filters and downsized the fully connected layers, resulting in the primary model introduced in this section. This allowed for efficient training while improving performance.

Another challenge was the imbalance in class distributions. Unlike the baseline model, which used oversampling, our CNN was trained on the raw imbalanced data. As a result, some piece types with fewer training examples (kings and queens) saw lower accuracy. Introducing focal loss, class-weighted loss, or balanced batch sampling may improve this in future iterations.

6 LINKS TO GITHUB AND COLAB NOTEBOOKS

- **Baseline Model:** `https://colab.research.google.com/drive/1xycC-Rng8AAzhwBufXHzHGLuvcElcc2S?usp=sharing`
- **Primary Model:** `https://colab.research.google.com/drive/1X_6f3QCK5oZ73d2Pce3HCc1nok8EPjDH?usp=sharing`
- **Github Link:** `https://github.com/Luthiraa/CVChess`

REFERENCES

ChessWatch. Chess events dynamics 2020–24. `https://chesswatch.com/news/chess-events-dynamics-2020-24`, 2024. Accessed: 2025-06-12.

Jacky Ding et al. Chessvision: Chess board and piece recognition. `https://web.stanford.edu/class/cs231a/prev_projects_2016/CS_231A_Final_Report.pdf`, 2016. Stanford CS231A Final Report.