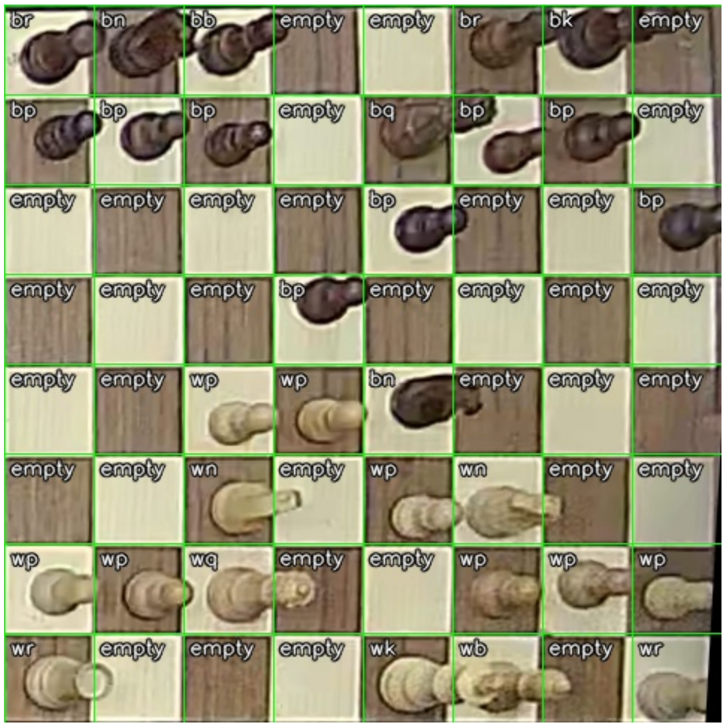
Chessboard Vision-Deep Learning

Noam Edeny

Ori Weiss

Shani Berechys


br bn bb empty empty br bk empty
bp bp bp empty bq bp bp empty
empty empty empty empty bp empty empty bp
empty empty empty bp empty empty empty empty
empty empty wp wp bn empty empty empty
empty empty wn empty wp wn empty empty
wp wp wq empty empty wp wp wp
wr empty empty empty wk wb empty wr

Final Project Report

January 13, 2026

Table of Contents

1. Abstract

This project focuses on reconstructing the state of a chessboard from a single real-world image. The task is defined as per-square classification, where each of the 64 squares is assigned to a chess piece class, an empty square, or an unknown class in the presence of occlusions. The predicted square labels are then aggregated to produce a full board representation in FEN (Forsyth–Edwards Notation) format.

We propose an end-to-end pipeline that extracts square-level image patches with spatial context and applies a pretrained ResNet-based classifier trained with a weighted cross-entropy loss to address class imbalance. Training stability and generalization are improved using early stopping, and out-of-distribution (OOD) handling is applied to identify uncertain or occluded squares.

The final model achieves 99.52% square-level accuracy and 98.64% accuracy on non-empty squares, demonstrating robust performance under real-world imaging conditions.

GitHub repository: chessboard-vision-ml

---

2. Introduction

2.1. Task description and motivation

The goal of this project is to classify each of the 64 squares of a chessboard from real images into one of the possible piece classes (e.g., white pawn, black knight, empty square, etc.). Based on the per-square predictions, the system is required to reconstruct the full board state and produce a standard digital representation in FEN (Forsyth–Edwards Notation) format.
The solution is based on processing a single static image, while handling occlusions and identifying cases in which the content of a square cannot be determined with certainty.

2.2. Challenges and goals

Beyond the basic difficulty of image classification, the project introduces several significant challenges:

- **Handling occlusions:** Identifying squares that are partially or fully occluded by a human hand or other pieces, and labeling them as "unknown" without harming the accuracy of the rest of the board.
- **Temporal information limitation:** The classifier must rely on a single image only, without using information from previous or subsequent frames during decision making.

- **Real-world conditions:** Dealing with images captured under varying camera angles, shadows, non-uniform lighting, and physical differences between chess piece sets.
- **Accuracy required for FEN reconstruction:** A single square misclassification can lead to an illegal or incorrect board reconstruction, which requires a high level of robustness from the model.

2.3. Main contributions

In this project, we propose:

- A **full end-to-end pipeline**: board image input → extraction of 64 squares → per-square classification → FEN assembly and generation of a matching board diagram.
- **Modular model design:** A generic architecture with interchangeable backbones (e.g., ResNet family), enabling consistent experimentation and comparison.
- **Spatial image processing with per-square prediction** rather than whole-board classification.
- **Chess-aware loss functions** designed to handle the natural piece distribution in chess games.
- **Systematic handling of uncertainty and occlusions:** Introducing a decision component that outputs "unknown" for squares suspected as OOD/occluded (as part of pre- or post-processing), in accordance with the project requirements.

---

3. Related Work

3.1. Relevant papers and datasets

- Lecture 5 – ResNet
- Practical Session 6 – ResNet and Early Stopping
- Practical Session 8 – Triplet Loss
- The labeled dataset provided as part of the project

3.2. Differences from prior work

Our work combines geometric decomposition of the problem using domain knowledge (chess) with a ResNet-based model and a task-adapted loss function.
Unlike approaches that rely on extensive data augmentation or large-scale external training infrastructures, we worked exclusively with the labeled dataset provided within the project and developed our own solution pipeline and experimental setup.

---

4. Method

4.1. Model architecture

The final model selected is a **pretrained ResNet** used as a backbone for feature extraction, combined with a small classification head adapted to 13 classes. In practice:

- The ResNet serves as a **feature extractor**.
- The original classification layer is removed or disabled, and a new head is built on top of the extracted features, consisting of a fully connected layer, an activation function, a Dropout component, and an output layer producing logits for 13 classes.

This choice was made after a series of experiments and comparisons. We tested different Dropout probabilities as well as a Multi-Learner Autoencoder architecture (combining reconstruction and classification), but these did not improve performance compared to a focused ResNet with appropriate loss weighting (see also the “What Did Not Work” / ablation sections later in the report).

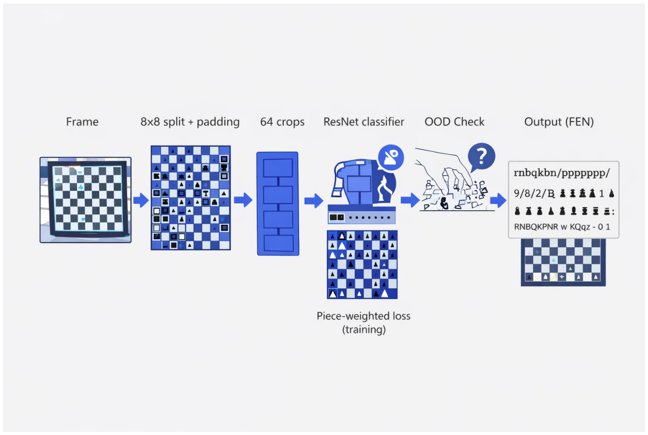


Figure 1: Model architecture

4.2. Input/output representation

Input:
We split the input (a chessboard image with 64 squares) into 64 separate square images. Since the images come from real games and are not necessarily captured from a perfectly top-down viewpoint, we added a safety margin around each square to account for cases where a piece slightly crosses square boundaries in the image.
The basic prediction unit is a single square. For each square, we extract an RGB patch that includes spatial context (padding/context) around the square boundaries (as described in the experiments section), resize it to a fixed resolution, and convert it to a tensor.

Output:
The model produces a multi-class prediction for each square patch among 13 classes (12 chess pieces by color + empty). At the frame level, the 64 predictions are aggregated into an 8×8 matrix and converted into a FEN string by compressing sequences of empty squares and mapping predictions to standard chess piece symbols.

4.3. Training procedure

Training was performed using a standard PyTorch training loop:

- Training on mini-batches of square patches (as described in the experiments section).
- Optimization using Adam with hyperparameters defined in an experimental configuration (e.g., learning rate, batch size, number of epochs, image size).
- Performance monitoring on a validation set throughout training, including Train vs. Validation curves (loss/metrics) to clearly detect overfitting.

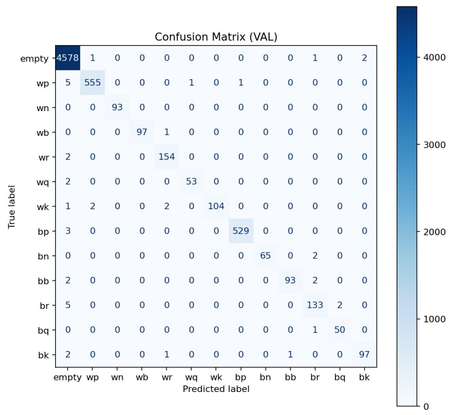
We used **Early Stopping** based on validation loss (with patience and min_delta) and restored the best-performing weights to reduce overfitting and avoid excessive training.

4.4. Loss functions

The base loss for the task is multi-class Cross-Entropy. However, since the *empty* class and pawns are much more frequent, we used **Weighted Cross-Entropy**:

- Higher weights were assigned to errors involving piece classes (especially "piece → empty" mistakes or confusions between pieces).
- A lower weight was assigned to errors in the empty class.

The weighting was chosen empirically to align the optimization objective with evaluation metrics focused on piece recognition (as described in the experiments section), so that the model is not mainly "rewarded" for correctly identifying empty squares.



	empty	wp	wn	wb	wr	wq	wk	bp	bn	bb	br	bq	bk
empty	4578	1	0	0	0	0	0	0	0	0	1	0	2
wp	5	555	0	0	0	1	0	1	0	0	0	0	0
wn	0	0	93	0	0	0	0	0	0	0	0	0	0
wb	0	0	0	97	1	0	0	0	0	0	0	0	0
wr	2	0	0	0	154	0	0	0	0	0	0	0	0
wq	2	0	0	0	0	53	0	0	0	0	0	0	0
wk	1	2	0	0	2	0	104	0	0	0	0	0	0
bp	3	0	0	0	0	0	0	529	0	0	0	0	0
bn	0	0	0	0	0	0	0	0	65	0	2	0	0
bb	2	0	0	0	0	0	0	0	0	93	2	0	0
br	5	0	0	0	0	0	0	0	0	0	133	2	0
bq	0	0	0	0	0	0	0	0	0	0	1	50	0
bk	2	0	0	0	1	0	0	0	0	1	0	0	97

Figure 2: Confusion matrix of ResNet (Actual labels Vs predicted)

In code:

```
nn.CrossEntropyLoss(
    weight=torch.tensor(
        [0.2] + [1.0] * 12,
        dtype=torch.float
    )
)
```

Formally:
Note that in our setup, class 0 corresponds to an empty square.

- $z = (z_0, \dots, z_{12})$ model logits
- Softmax:

$$p_i = \frac{e^{z_i}}{\sum_{j=0}^{12} e^{z_j}}$$

- Weight vector:

$$w = (w_0, \dots, w_{12}), w_0 = 0.2,\ w_i = 1\ \forall i \geq 1$$

- Loss:

$$L(y, z) = -w_y \cdot \log(p_y)$$

where y is the ground-truth label, $z$are the logits, and $w_y$is the weight of the corresponding class.

4.5. Preprocessing / Postprocessing

Preprocessing:
Includes splitting the board image into squares, extracting square patches with context, resizing, and normalization to tensors.

Postprocessing:
We incorporated an Out-of-Distribution (OOD) mechanism to identify squares for which the model's predictions are unreliable and should therefore be marked as unknown, for example in cases where squares are occluded by a player's hand or head. During this process, we observed that using a single global OOD threshold was insufficient. Since the loss function explicitly differentiates between empty squares and squares containing a piece, the confidence distributions for these two cases differ significantly. Consequently, we applied class-dependent OOD thresholds: an optimal threshold of 0.5 for squares predicted to contain a piece, and a higher threshold of 0.75 for squares predicted as empty.

5. Experiments

5.1. Dataset description and splits

All experiments were conducted using only the labeled dataset provided as part of the project. Each frame-level example consists of a full board image and a board-state label in FEN format. From each frame, we generated square-level data, i.e., 64 examples per frame (one image patch per square with a corresponding label).
The model is trained and evaluated at the square level, while still allowing evaluation at the board/image level.

Train/Validation/Test splits were performed at the **frame level** to reduce information leakage between sets, since squares from the same frame and frames from the same game are highly correlated. The split was defined as 60% Train, 20% Validation, and 20% Test. After splitting, each set was expanded to square-level data for training and evaluation.

Square patches are loaded using a DataLoader and trained in mini-batches, where each batch contains patches from different squares (and sometimes different frames), enabling stable optimization and efficient hardware utilization.

5.2. Evaluation metrics

Performance evaluation in this project requires care, since the model can achieve relatively high scores by correctly classifying empty squares, which does not necessarily reflect the quality of chess piece recognition. Accordingly, we used several complementary metrics:

- **Square-level Accuracy:** Percentage of correctly classified squares over all squares. This provides a general performance indicator but can be biased by the prevalence of empty squares.
- **Image/Board-level Metric:** Aggregates the 64 predictions of a single frame into a single score (e.g., average correct squares per frame and/or a "perfect board" criterion where all 64 squares are correct). This better represents the application goal of reconstructing a full board state.
- **Non-empty / Piece-focused Accuracy:** Accuracy or error computed only over squares that are non-empty in the ground truth, directly evaluating the quality of piece and color recognition while reducing bias from empty squares.

In addition to numerical metrics, we generated confusion matrices to analyze class confusions, as well as Train vs. Validation curves (loss and/or accuracy over epochs) to visually and clearly identify overfitting.

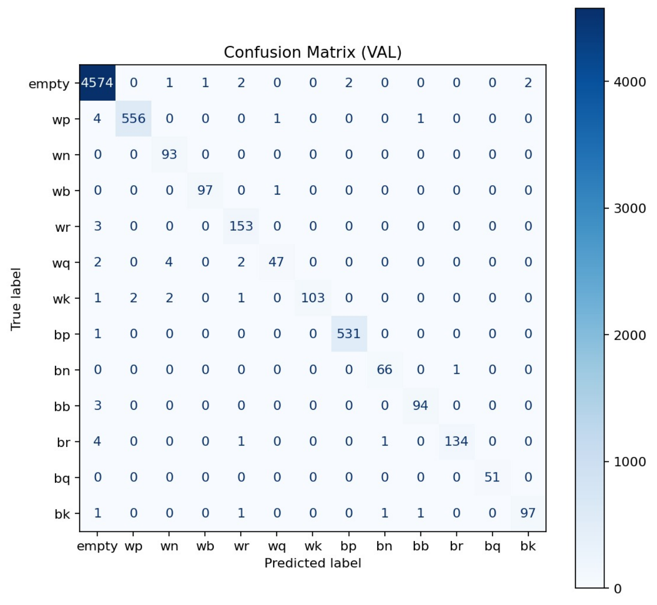
5.3. Quantitative results

Model	***Backbone***	***Weighted Loss***	***Dropout***	***AE Component (Reconstruction)***	***Test Acc (All)***	***Test Acc (No-Empty / Pieces)***
CNN	CNN	✗	–	✗	0.9836	0.9613
CNN Weighted	CNN	✓	–	✗	0.9866	0.9637
ResNet	ResNet (pretrained)	✗	–	✗	0.9946	0.9831
ResNet Weighted (Final)	ResNet (pretrained)	✓	–	✗	0.9952	0.9864
ResNet Weighted + Dropout 0.1	ResNet (pretrained)	✓	0.1	✗	0.9898	0.9705
ResNet Weighted + Dropout 0.3	ResNet (pretrained)	✓	0.3	✗	0.9884	0.9806
ResNet Weighted + Dropout 0.5	ResNet (pretrained)	✓	0.5	✗	0.9922	0.9806
Multi-Learner Autoencoder Weighted	Encoder–Decoder + Classifier	✓	–	✓	0.9311	0.8109

The table summarizes the test results for all configurations. For fairness, we emphasize the **Test Accuracy (No-Empty / Only Pieces)** metric, which neutralizes the bias introduced by easy success on empty squares and directly measures piece recognition quality.

5.4. Final model selection:

ResNet Weighted was chosen, as it achieves the best overall accuracy and—most importantly—the highest piece-focused accuracy (0.9864 on No-Empty / Pieces).



	empty	wp	wn	wb	wr	wq	wk	bp	bn	bb	br	bq	bk
empty	4574	0	1	1	2	0	0	2	0	0	0	0	2
wp	4	556	0	0	0	1	0	0	0	1	0	0	0
wn	0	0	93	0	0	0	0	0	0	0	0	0	0
wb	0	0	0	97	0	1	0	0	0	0	0	0	0
wr	3	0	0	0	153	0	0	0	0	0	0	0	0
wq	2	0	4	0	2	47	0	0	0	0	0	0	0
wk	1	2	2	0	1	0	103	0	0	0	0	0	0
bp	1	0	0	0	0	0	0	531	0	0	0	0	0
bn	0	0	0	0	0	0	0	0	66	0	1	0	0
bb	3	0	0	0	0	0	0	0	0	94	0	0	0
br	4	0	0	0	1	0	0	0	1	0	134	0	0
bq	0	0	0	0	0	0	0	0	0	0	0	51	0
bk	1	0	0	0	1	0	0	0	1	1	0	0	97

Figure 3: Confusion matrix of ResNet Weighted (Actual labels Vs predicted)

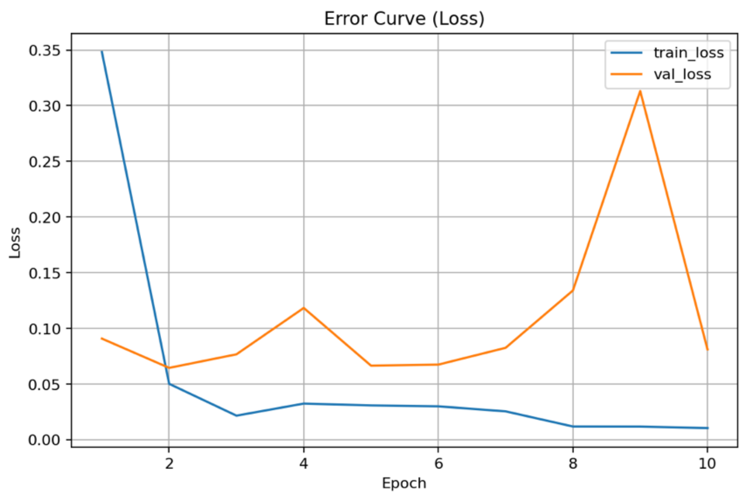


Figure 4: ResNet Weighted loss curve

For configurations with Dropout and the Multi-Learner Autoencoder, saved artifacts (confusion matrices and metric plots) are included and referenced in the appendix or main report as complementary qualitative summaries.

6. Ablation Study

We conducted ablations to justify the components of the method and examine which parts contribute in practice.

6.1. Backbone effect: CNN vs. ResNet

Transitioning from a basic CNN to a pretrained ResNet consistently improved performance:

- **CNN → ResNet (unweighted):**
  - Test Acc (All): 0.9836 → 0.9946
  - Test Acc (No-Empty): 0.9613 → 0.9831

Conclusion: A deeper and stronger backbone (ResNet) significantly improves generalization and piece recognition accuracy.

6.2. Effect of weighted loss (stronger penalty on piece errors)

Adding class weights improved piece-focused metrics for both CNN and ResNet:

- CNN: No-Empty 0.9613 → 0.9637 (+0.0024)
- ResNet: No-Empty 0.9831 → 0.9864 (+0.0034)

Conclusion: Loss weighting aligns the optimization objective with the research/application goal (piece recognition) and reduces bias from the empty class.

6.3. Effect of Dropout (under weighted loss)

We tested different Dropout probabilities (0.1, 0.3, 0.5). Compared to ResNet Weighted without Dropout:

- No-Empty accuracy dropped from 0.9864 to 0.9806 (p=0.3/0.5) and to 0.9705 (p=0.1).

Conclusion: In this setting, Dropout did not improve performance and in some cases degraded it, and therefore was not included in the final model.

6.4. Effect of Multi-Learner Autoencoder

Combining reconstruction with classification led to a sharp performance drop:

- Test Acc (No-Empty): 0.9864 (ResNet Weighted) vs. 0.8109 (ML-AE Weighted).

Conclusion: The reconstruction objective likely competes with the discriminative classification objective, making it unsuitable for the current problem formulation and dataset scale.

Main ablation conclusion:
The most effective combination is:
Pretrained ResNet + Weighted Loss, without Dropout and without a reconstruction component.
This setup provides the best balance between overall accuracy and true accuracy on non-empty squares, which is the critical metric for the project.

---

7. What Did Not Work

During the project, we explored several directions aimed at improving generalization and piece recognition, especially under class imbalance between empty and occupied squares. Some approaches did not improve performance (and sometimes harmed it), and were therefore abandoned.

1. **Adding Dropout at different rates to the weighted ResNet model**
   We experimented with Dropout probabilities (e.g., 0.1 / 0.3 / 0.5), assuming it would reduce overfitting. In practice, compared to the weighted ResNet without Dropout, these configurations did not yield consistent improvements and even reduced performance on the key metric (non-empty accuracy). Therefore, Dropout was excluded from the final model.
2. **Multi-Learner Autoencoder (Reconstruction + Classification)**
   We tested an architecture combining a reconstruction objective with a discriminative classification objective, expecting shared representation learning to improve generalization. In practice, performance dropped significantly, especially for piece recognition. Our conclusion is that the reconstruction objective competes with classification by forcing the model to explain pixel-level details that are not necessarily useful for distinguishing chess pieces, which is particularly harmful under limited labeled data.
3. **Vision Transformers**
   We studied Vision Transformer-based models in depth as part of the literature review and course materials and considered integrating them. However, at an advanced planning stage, we decided not to implement them, as we estimated that they are not well-suited to the available labeled data size and involve relatively high experimental cost compared to pretrained CNN/ResNet models.
4. **Training without class weighting (unweighted loss)**
   Early experiments used unweighted loss. Although overall accuracy was high, it was biased by the prevalence of empty squares, while piece recognition did not improve accordingly. Switching to weighted loss that penalizes piece errors more heavily led to consistent improvements in piece-focused metrics and was retained in the final model.

Key insights:
Negative experiments highlighted two main points:
(1) Under limited and imbalanced data, not every “standard” regularization technique is beneficial in practice;
(2) Auxiliary objectives that seem theoretically promising (e.g., reconstruction) may distract learning from the core discriminative goal.
Accordingly, the final solution was preferred due to its relative simplicity, good fit to the data, and consistent improvement on metrics that truly reflect the project’s objectives.

---

8. References

1. Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). **Dropout: A simple way to prevent neural networks from overfitting**. *Journal of Machine Learning Research, 15*, 1929–1958.
2. Wager, S., Wang, S., & Liang, P. S. (2013). **Dropout training as adaptive regularization**. In *Advances in Neural Information Processing Systems (NeurIPS)*.
3. He, K., Zhang, X., Ren, S., & Sun, J. (2016). **Deep residual learning for image recognition**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770–778).
4. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). **An image is worth 16×16 words: Transformers for image recognition at scale**. In *International Conference on Learning Representations (ICLR)*.
5. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). **Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion**. *Journal of Machine Learning Research, 11*, 3371–3408.
6. Stanford CS231n. (n.d.). **Neural Networks Part 2: Setting up the data and the loss (Regularization section)**. *CS231n: Convolutional Neural Networks for Visual Recognition*. Retrieved January 8, 2026, from the CS231n course website.
7. PyTorch. (n.d.). **torch.nn.Dropout**. *PyTorch Documentation*. Retrieved January 8, 2026, from the official PyTorch documentation.
8. Course Staff. (2026). **Practical Session 7,9** (PDF handout). Unpublished course materials.