Machine Learning for Chess Movement Recognition

Youssef Boughizane, Cléo Renaud, Sofia Taouhid
CS-433, EPFL, Switzerland

***Abstract*—This project aims to automate the digitization of handwritten chess moves from scoresheets using an open-source model that operates locally on ChessReader servers, replacing the current reliance on a paid third-party OCR engine API. Our contributions are two-fold: first, we create a custom dataset to address the data scarcity issue; second, we explore two model training strategies: training custom CNN-RNN models from scratch and fine-tuning existing pretrained models. We train these models both with and without our custom dataset and demonstrate its contribution to enhancing model performance.**

I. Introduction

Chess players in Over-the-Board events use scoresheets to record their moves by hand, which are later digitized by event organizers for official records. Additionally, many players wish to digitize their past games for archival and review purposes. The ChessReader web application was developed to address these challenges. Currently, the app relies on a paid, third-party OCR engine API to recognize and transcribe these handwritten moves. While this solution is effective, it introduces dependencies on external services, limiting scalability and control over data. Therefore, our goal is to develop open-source models that can perform this task with similar accuracy, providing greater autonomy and flexibility. In the following sections, we will first define our task more precisely and describe the datasets we used to train our models (II). We will then provide an overview of these models (III), and finally, we will report our results (IV).

II. Context

Before precisely defining the task, let's first introduce the *Chess algebraic notation* :

Chess algebraic notation is a standardized system for recording moves. Each square is identified by coordinates ('a'–'h' for columns and '1'–'8' for rows). Pieces are represented by letters: 'K' (King), 'Q' (Queen), 'R' (Rook), 'N' (Knight), and 'B' (Bishop), while pawns are implicit. Moves include the destination square and optional symbols: 'x' for captures (e.g., Bxe4), '=' for promotions (e.g., e8=Q), '+' for check, and '#' for checkmate.

The ChessReader app already segments the scoresheets into individual boxes, each containing a single move. In the context of this project, we treat each box as an input, with the goal of predicting the corresponding algebraic notation for the move it contains. This process is illustrated in Figure 1.

A. Datasets

We used the following datasets to train and evaluate our models:

1) **Chess Reader dataset**: This dataset was provided to us by the professor, it contains $\approx$ 1600 images of chess moves, some of which are in French or German and some don't have human validated labels. Once we eliminate these, we are left with ≈ 500 datapoints (image moves + labels).
2) **Handwritten Chess Scoresheet (HCS) dataset** [1]: This open source dataset consists of $\approx$ 13k images of tightly cropped boxes containing chess moves and their correspond labels. All games are written in English notation.
3) Our **Custom Dataset**: We created a dataset with $\approx$ 4k image box by handwriting ourselves publicly available chess games. More details on this dataset can be found in the following section.

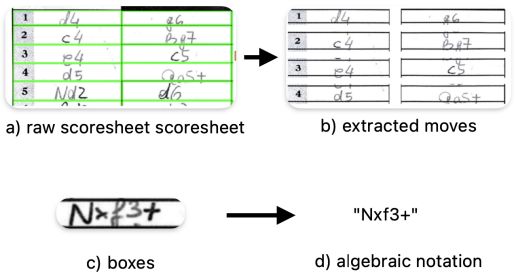

Fig. 1. Above is the transformation already performed by ChessReader, while below is the task we aim to accomplish in this project.

B. Data Creation

A significant challenge in this project was the limited amount of data available in the provided dataset. Initially, we used the publicly available HCS dataset [1], but further improvements to our models were constrained by its size.

Although a previous group attempted to generate synthetic "handwriting" of chess moves, their models quickly overfitted to the synthetic data. To address this, we focused on collecting real handwritten data.

While one option was to collect, scan, segment move boxes, and manually label scoresheets, this was going to be too expensive time wise.

Instead, we manually created additional handwritten chess scoresheets by sourcing Portable Game Notation (PGN) files of chess games from an online dataset [2], transcribing the moves by hand, and inviting friends and acquaintances to contribute their handwriting for diversity. These sheets were then scanned to produce a more varied and representative dataset. To enhance dataset diversity, we incorporated variations in writing styles, pen colors, pressure, letter size, spacing, and alignment within the designated boxes.

To simulate real-world conditions, we also included examples with crossed-out moves and corrections using white-out. These enhancements captured the natural variability

of handwritten scoresheets, improving the robustness and generalization capability of our model.

We provide this dataset in our github repository.

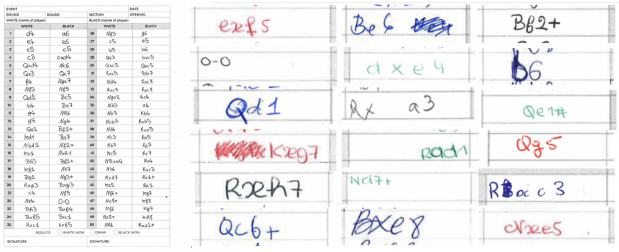

Fig. 2. Custom Dataset Sample

C. Data pre-processing

To prepare our datasets for training, we applied the following preprocessing and augmentation techniques:

- **Shuffling:** Randomized the order of data points.
- **Resizing:** Resized all images to 64 x 256 to ensure compatibility with the CNN architecture.
- **Data Augmentation:** Applied the following standard augmentation techniques to the training data:
  - Random brightness adjustment
  - Random erosion and dilation
  - Random sharpening
  - Random rotation (with an angle up to 10 degrees, replacing newly created pixels due to rotation with the median of the image)

These steps help improve the model's robustness, prevent overfitting, and ensure the data is well-structured for efficient training.

D. Data splitting

A realistic use case for our model involves evaluating its ability to predict chess moves from unseen handwriting styles and formats, such as variations in brightness, paper type, and cropping. To simulate this scenario, we carefully split the datasets as described in Figure 3. The resulting distribution is the following :

- **Test set:** 100% of ChessReader, which is provided by the professor, ensuring that the model is evaluated on handwriting styles and formats it has never seen during training.
- **Training set:** 25.4% from Custom, 71.8% from HCS, and 2.7% from ChessReader.
- **Validation set:** 20.4% from Custom, 57.7% from HCS [to ensure that the validation distribution loosely matches the test set], and 21.9% from ChessReader [to help mitigate overfitting and choose a model that generalizes].

This is illustrated in Figure 3.

III. Models and Methods

A. Methods

In this section, we provide a general introduction to the techniques underlying our models. We begin by introducing the CNN-RNN framework for handwritten recognition, along with the loss function used for training, namely the CTC loss. We then present an overview of the architecture of the model we fine-tune.

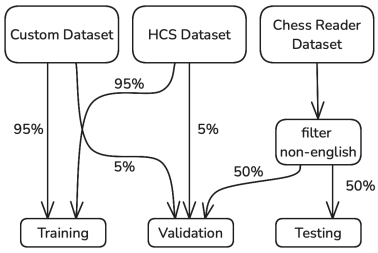

Fig. 3. Data Repartition

CNN and Bidirectional Long Short-Term Memory (CNN-BiLSTM): The CNN-BiLSTM architecture combines convolutional layers for feature extraction and capturing local visual patterns, followed by RNN layers which output a new sequence of features while taking into account the context ("mixing information across feature maps") and handling variable-length inputs ("d4" vs. "Qxf6"). The LSTM, a specialized RNN, mitigates the vanishing gradient problem, while the BiLSTM improves performance processes data in both directions.

Connectionist Temporal Classification (CTC) Loss: It is a sequence alignment loss function commonly used for tasks with variable-length input and output sequences that lack explicit alignment. For instance, in our context, predictions such as 'd44', 'dd4', or 'd[blank]4' are all considered valid alignments of the target sequence 'd4'. CTC loss accounts for this by summing the probabilities of all valid alignments when computing the negative log-probability of the target sequence.

Transformer-based encoder-decoder: The CNN-RNN approach can be seen as an encoder-decoder, with the CNN extracting features and the RNN generating text predictions. In contrast, recent state-of-the-art models use a transformer-based encoder-decoder, where the encoder (e.g., Vision Transformer) splits the input image into patches and processes them into embeddings, while the decoder autoregressively generates text predictions from the sequence of embeddings.

Metrics: **Character Error Rate (CER)** The Character Error Rate (CER) measures the ratio of incorrectly predicted characters, where lower values indicate better performance. It is computed as:

$$CER = \frac{S + D + I}{N}$$

where S is the number of substitutions, D the deletions, I the insertions, and N the total number of characters.

Word Error Rate (WER) evaluates the accuracy of text recognition systems and is given by:

$$WER = 1 - Accuracy = \frac{\text{\# of incorrect predictions}}{\text{total \# of predictions}}$$

B. Models

We tested three models:

*1) **CNN-BiLSTM from the MLTU[3] library:*** The architecture consists of nine convolutional residual blocks (reducing spatial dimensions and increasing feature depth

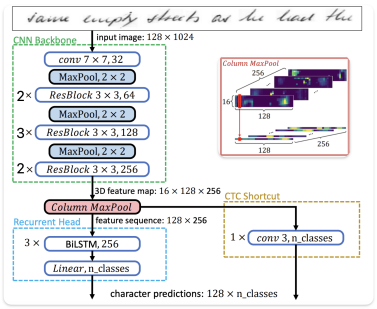

Fig. 4. HTR-NET model architecture

from 3 to 64 channels). Each block includes batch normalization, LeakyReLU activation, and residual connections. A bidirectional LSTM (BiLSTM) with 128 hidden units per direction (256 total) follows, along with a Dropout layer. The BiLSTM output goes through a linear linear which are mapped to a probability distribution over characters logits for CTC loss computation, and a log-softmax function provides probabilities.

*2) **HTR-NET[4]** architecture:* This model adapts the CNN-BiLSTM architecture by increasing convolutional channels and using column-wise max pooling instead of concatenation. A shortcut branch with 1D convolution enables parallel character sequence estimation, forcing the CNN backbone to produce more discriminative outputs. The model also uses 3 BiLSTM layers instead of 1. See figure 4.

*3) **TrOCR (Transformer OCR)**:* Developed by Microsoft Research, this model uses a vision transformer (ViT) encoder and a text transformer decoder for OCR. The 'microsoft/trocr-small-handwritten' variant, fine-tuned on the IAM Handwriting Database, uses cross-entropy loss and has 61 million parameters. Its architecture is based on DeiT and UniLM, enabling efficient recognition of printed and handwritten text.

For the models trained from scratch, we use the MLTU library [3], which provides implementations for data loading, augmentation, and model training. It also integrates TensorBoard to track key metrics like CER, WER, and training progress, helping prevent overfitting. This allowed us to focus on model architecture and experimentation rather than re-implementing common utilities.

IV. Results

We initially used the CNN-BiLSTM model from the MLTU library. We started training the model giving it images of size 32x128. This led to a Character Error Rate (CER) converging to 0.155 after 150 epochs. We then modified two things. A first change was to increase the image size to 64x256. The second change was on the approach used for transformations in data augmentation. Initially it was done by filling new pixels with a random value, but we decided to make it fill the new pixels with the median of initial pixels. These two adjustments helped us make the CER decrease. However, we tried increasing the image size further, but it did not yield any additional improvement. Consequently, by improving our preprocessing methods we achieved an increase in accuracy.

The next step we considered was to add more data, which is the reason we created our own custom dataset that we introduced earlier in the report. With this additional data (around 4k more moves), the model's performance increased (see Tables I and II). Therefore, the model generalizes better when we add data. However, it required many epochs to converge. This suggested that the CNN-BiLSTM model might not be complex enough for this task.

This led us to explore a larger model, HTRNet, introduced previously. It is more complex due to a higher number of parameters and modules. We first trained it without using our customized dataset. When we added our custom dataset, the model performance improve, demonstrating a better generalization to the test set when additional data is given. The complexity of the model also ensures a faster convergence which suggests that this model could be better to use for this task.

These findings suggest that with even more data, the performance of the two models could improve further. Unfortunately, we did not have enough time to increase the size of our dataset any further to verify our assumption.

Custom Dataset	**WITHOUT**		**WITH**	
	Val	**Test**	**Val**	**Test**
CNN-BiLSTM-MLTU	0.1581	**0.3559**	0.0948	**0.1761**
HTR-Net	**0.152**	0.3601	**0.091**	0.2021

TABLE I
MODEL PERFORMANCE IN TERMS OF **CER**.

Custom Dataset	**WITHOUT**		**WITH**	
	Val	**Test**	**Val**	**Test**
CNN-BiLSTM-MLTU	0.3033	**0.5506**	**0.195**	**0.3279**
HTR-Net	**0.29**	0.6073	0.2004	0.3846

TABLE II
MODEL PERFORMANCE IN TERMS OF **WER**

Our results, summarized in Tables I and II, compare the two configurations with and without the inclusion of the Custom dataset (the one we created ourselves).

For both models, we observe a significant decrease in the validation CER and WER when our dataset is included.

The difference in test CER and WER is even more pronounced, demonstrating that our dataset reduces overfitting and improves generalization. This highlights the quality and utility of the dataset we created.

We also observe that CNN-BiLSTM-MLTU outperforms across all metrics, particularly during testing, indicating that it generalizes better due to its lower complexity. However, the impact of the new dataset is even more apparent for HTR-NET, which is a more complex model. This suggests that HTR-NET could still be improved significantly with additional training data.

Additionally, we fine-tuned the TrOCR model using the training set as defined above and report the results in table III. While the TrOCR model performs slightly worse than the other models, it remains competitive overall, demonstrating its effectiveness in recognizing handwritten data.

Metric	Validation Set	Test Set
CER	0.1044	0.1814
WER	0.2193	0.3482

TABLE III
VALIDATION AND TEST SET RESULTS FOR TROCR FINE-TUNING.

And as a final step, we implemented majority voting between our three models, resulting in a model called MIX. In the case of a tie, we select the prediction of the CNN-BiLSTM-MLTU, which is the best model based by CER on the validation set described above. MIX is the best performing model overall both in validation and test with 0.05 and 0.11 of CER and WER respectively during validation. The results on the test set are provided in table IV.

Upon further inspection, we observed that some test images are of poor quality (unreadable and very loosely cropped). Although we kept these images in the test set, removing them would also have been a reasonable choice, as they are potentially unreadable even for humans. This would likely have resulted in even more impressive outcomes.

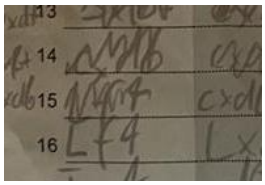

Fig. 5. Example an ambiguous test image. This should be cxd6. We can see xd6 if we squint.

Comparison with other OCR engines

As we were initially provided with the predictions of different OCR engines on the test set, we computed the accuracy of each of them in the following table and compare them with our methods.

TABLE IV
CER AND WER RESULTS ON TEST SET

Model	CER	WER
Third Party OCR		
Google	0.2541	0.4413
Azure	0.2877	0.3765
Abbyy	0.2548	0.3765
Our Models		
CNN-BiLSTM-MLTU	0.1761	0.3279
HTR-Net	0.2021	0.3846
TrOCR	0.1814	0.3481
MIX	**0.1718**	**0.2914**

Among the third-party OCR systems, Abbyy delivers the best performance.

Among our models, the CNN-BiLSTM-MLTU leads with a CER of 0.1761 and a WER of 0.32, followed by the fine-tuned TrOCR, and then the HTR-Net. A mixture of these three models outperforms each one individually, significantly improving on third-party OCR systems. This suggests that each model is capturing unique and valuable insights.

These results highlight the effectiveness of our models, demonstrating a notable reduction in error rates compared to third-party alternatives. Additionally, expanding our dataset with more data could further enhance performance, yielding even more powerful results.

Finally, an easy improvement we did not implement due to time constraints involves selecting the highest probability *valid chess move* rather than the highest probability *string*. We anticipate that this adjustment could lead to further improvements, particularly in WER.

V. ETHICAL RISK

A. Risk Identification

One ethical risk we identified concerns expanding our dataset with handwritten chess moves from friends.

- **Who is impacted?** The primary stakeholders are the individuals who contributed their handwriting voluntarily.
- **What is the negative impact?** While we don't require identification of contributors, there is a risk of privacy violations if their data is not anonymized. Identifiable patterns could potentially link back to individuals, raising concerns about consent and anonymity.
- **How significant is the risk?** The likelihood of privacy issues is low due to our precautions, but failure to address anonymity or biases could harm the project's ethical standing and trustworthiness.

B. Risk Mitigation

To mitigate this risk, we focused on three key steps: **contributors' consent, data diversity, and privacy protection.**

We first asked contributors to explicitly give their informed consent to participate in the dataset creation. Secondly, we had to make sure to have a diverse set of contributors with varied handwriting styles to avoid dataset biases because of demographic fairness. And finally, we had to ensure all handwriting samples are anonymized, so that no personally identifiable information remains. This anonymization wasn't a barrier to our project as from a technical point of view, knowing the identity of a contributor doesn't serve the project's goals.

VI. CONCLUSION

From our study, we observed that our models generalize better when the dataset we created is included. This highlights the crucial role of data, emphasizing that both the quantity and quality of data, such as diverse handwriting styles, are key factors for achieving strong performance. This serves as an important lesson: in machine learning, the data we work with is just as important as the models we train.

Future work may also incorporate the context of the entire game, such as the moves played before and those that will be played after, could also significantly improve the accuracy of move recognition.

REFERENCES

[1] N. Majid and O. Eicher, "Digitization of Handwritten Chess Scoresheets with a BiLSTM Network," vol. 8, no. 2, p. 31. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8879196/

[2] IronicNinja, "Raw chess games (pgn)," 2024, accessed: 2024-06-17. [Online]. Available: https://www.kaggle.com/datasets/ironicninja/raw-chess-games-pgn

[3] PyLessons, “Machine learning training utilities,” https://pypi.org/project/mltu/.
[4] G. Retsinas, G. Sfikas, B. Gatos, and C. Nikou, “Best practices for a handwritten text recognition system,” 2024. [Online]. Available: https://arxiv.org/abs/2404.11339