Exercise 3

183.663 Deep Learning for Visual Computing

Christoph Bartmann

September 7, 2023

1 Introduction

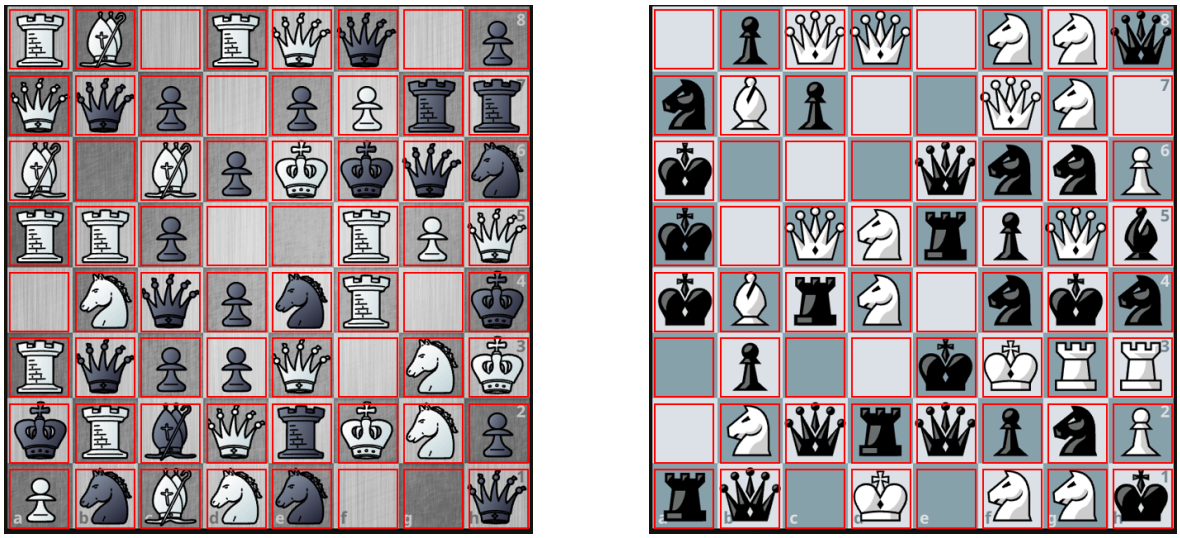
This project is centered on converting a snapshot of a 2D chess board into a string-based encoding, specifically the Forsyth-Edwards Notation (FEN). Our core approach involves using an object detection model, particularly YOLOv5, to accurately identify the chess pieces on the board along with their respective positions. The FEN system is a standardized notation utilized to depict a particular board setup during a game of chess. It encapsulates all crucial data required to resume a game from a specific point, including the arrangement of pieces on the board. This project was inspired by two main references, namely[3] and [1]. The first reference provides the methodology for dataset creation and project workflow, though its application does not employ object detection but instead divides the screenshot into tiles, each processed through a CNN. Conversely, the latter uses object detection models such as YOLOv5, but its focus is on real-world chess pieces rather than screenshots, as illustrated by the provided dataset [2].

2 Dataset Generation

The project's dataset was specifically created from the ground up, utilizing the strategy illustrated in [3]. Given that there are 12 unique chess pieces and 64 tiles, the possible configurations approximately reach up to 10^{69}. It is impractical to create a dataset representing all these combinations. Consequently, our approach incorporated even infeasible configurations, enabling the model to focus on the inherent details in each tile rather than relying solely on the relative positions of pieces. Nevertheless, given that empty tiles and pawns tend to be overrepresented in most scenarios, we infused a bias reflecting this reality in our random FEN generation. The use of infeasible settings also allows us to oversample the representation of each chess figure, which makes the dataset smaller than one featuring the same number of figure instances and only viable board positions.

Lichess, an online chess server, was the source of data for this project. It provides a RESTful API capable of rendering a chess board based on a specific FEN configuration. Using the endpoint 'https://lichess.org/editor/<FEN-STRING>', a page displaying a chess board as per the FEN string's configuration can be loaded. Automatic screenshot capturing of the board with the corresponding game state was realized with the Selenium framework. Lichess offers an array of board themes and chess piece styles which contribute to the diversity of our dataset. These features are incorporated into our data generation process via specific cookies, albeit extracting these cookies from the Lichess website can be quite laborious. In

this project, we made use of 14 distinct board themes and 9 chess piece styles to generate our images, despite Lichess offering a more extensive selection. Examples of two two training samples using different board and figure styles are shown in Figure 1.



(a) Randomly generated board setting with theme Metal and figure style Fantasy

(b) Randomly generated board setting with theme Blue and figure style Kosal

Figure 1: Two training samples using different board themes and figure styles. The red squares represent the ground truth bounding boxes used for training.

The use of varied board themes and piece styles during data generation enhances the variability in our dataset, contributing to the resilience and generalization capabilities of our trained model. The model is thus able to identify chess pieces across a broad spectrum of visual representations, boosting its accuracy in interpreting new and unseen images.

As our approach uses the same screenshot each time and the board configuration is known, the data can be preprocessed easily to be compatible with a YOLO framework. This annotation employs a consistent grid of bounding boxes for each image, maintaining the same bounding box irrespective of the piece (as seen in Figure 1). However, future iterations could consider more specific bounding boxes based on the exact chess piece.

Using the described pipeline, we generated a tailored dataset encompassing 2016 images. This can be understood as 16 instances of the corresponding mix of theme and figure style. Each board theme is thus represented via 144 images, and each figure style is depicted through 224 images respectively. The distribution of actual chess pieces throughout the entire dataset is illustrated in Figure 2, showcasing a fairly uniform pattern with an induced bias towards empty tiles and pawn pieces. We can also see that all pieces have at least 8000 instances across the dataset. Upon preparation, the dataset was divided into training, testing, and validation subsets, using a random split of 70%, 15%, and 15% respectively.

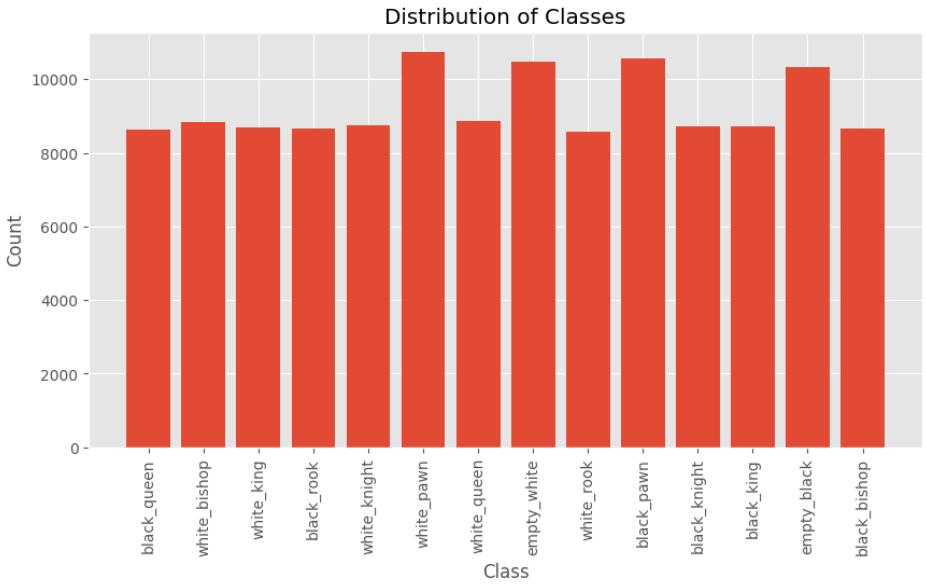


Figure 2: Distribution of Class Instances across the Dataset

3 Model Training

As previously highlighted, the primary objective of this project is to utilize deep learning based object detection via YOLOv5. Accordingly, we employed the repository released by Ultralytics [5] and their corresponding tutorial [4]. Our custom-made dataset was transformed into the required format and, leveraging the GPUs of Google Colab, the model was trained. We utilized the yolov5m model without much deliberation; As this is the third largest model with approximately 40 million parameters. For practical applications, it might be worthwhile to use a smaller model to achieve similar performance with reduced inference time. We implemented transfer learning using the available COCO weights. The bounding boxes used were the default ones, but during training, all layers were updated. The optimization strategy employed was ADAM with a learning rate of 1e-3. We also performed data augmentation, which included techniques such as horizontal and vertical flips and mosaic augmentation (all at 50%). For further details on the specific training parameters, refer to the *opt.yaml* file. The model was trained for 100 epochs and the results are depicted in Figure 3.

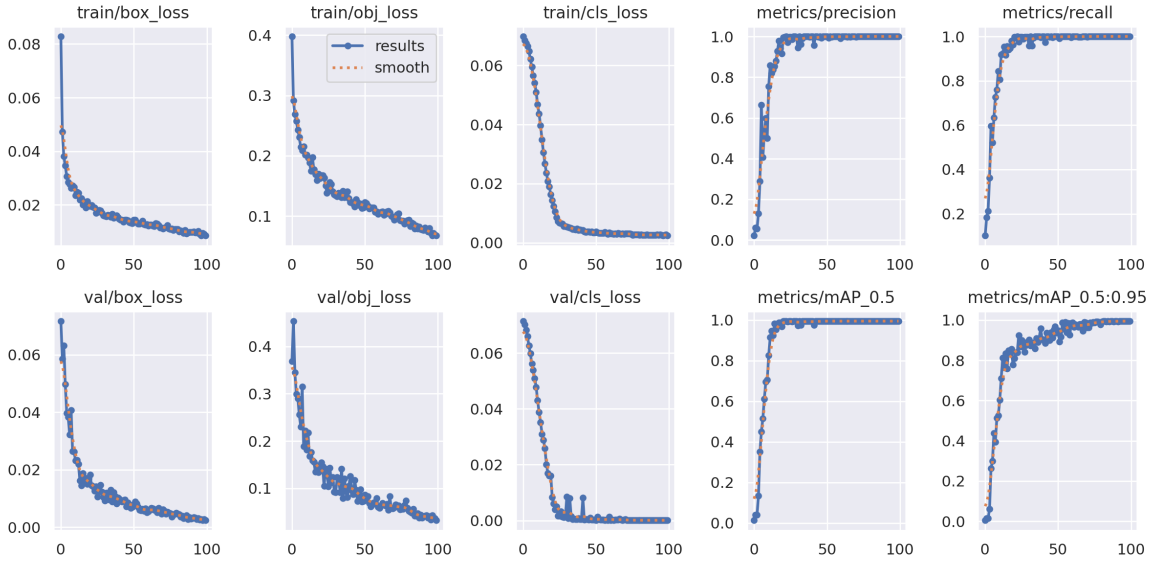


Figure 3: Training Metrics across 100 epochs

As seen from the results, the model already performs impressively after just a few epochs. The classification accuracy improves rapidly (Figure 3 cl_loss), while the bounding box error continues to decrease, albeit its numerical value remains very small. The obj_loss, which penalizes inaccuracies in predicting object presence and bounding box coordinates, shows steady decrease indicating room for some improvement. This encourages the model to accurately predict whether an object exists in a particular grid cell and to correctly localize the object within the bounding box. However, overall the training proceeded without any issues.

4 Results

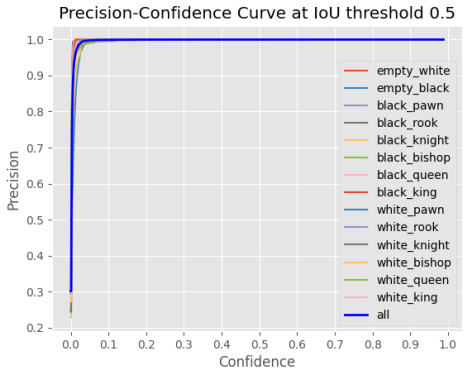
Utilizing the weights from our best-performing model, we performed inference on the training, validation, and test data sets. However, for the scope of this discussion, we will primarily concentrate on the results from the test data. The paths in the corresponding notebook can be easily adjusted for those interested in the other sets of data. Regrettably, we discovered that there is a dearth of general frameworks for the analysis of object detection data, leading us to build our own evaluation pipeline from scratch.

For this task, we considered several metrics, including Intersection over Union (IoU), Precision, Recall, F1-score, and, pertaining to our specific task, the accuracy of translating the image to the correct FEN string.

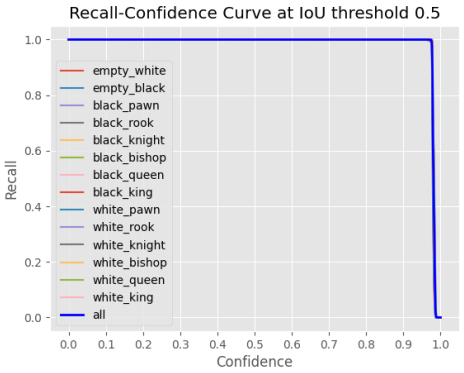
- Intersection over Union (IoU) is a measure of the overlap between the predicted bounding box and the ground truth. It ranges from 0 to 1, where 1 signifies a perfect match.
- Precision is the proportion of true positive results (correctly identified objects) among all positive results (all detected objects), indicating the accuracy of the object detector.
- Recall, or sensitivity, is the proportion of actual positives that are correctly identified, showing how well the model can find all the relevant cases within the dataset.
- The F1-score is the harmonic mean of precision and recall, providing a balanced measure when the distribution of positives and negatives is uneven.

The model's output encompasses the confidence level for each prediction. The analysis in this case is not as straightforward as in a classification scenario, as for each image, we predict both the bounding boxes and the class labels associated with a certain degree of confidence. Based on the confidence level and a fixed IoU threshold value of 0.5, we derive the results shown in Figure 4.

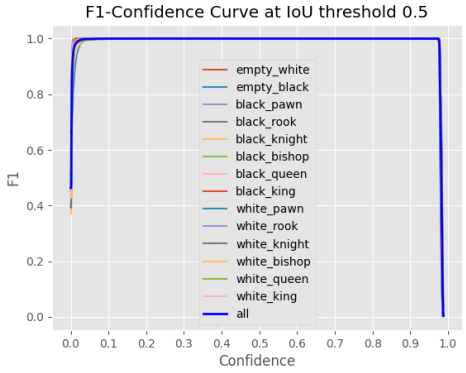
The model exhibits exceptional performance across all metrics, a result that aligns with the training outcomes. Considering that the training and test data are virtually identical, lacking significant variation, the superior performance of the model is understandable. Intriguingly, a similar performance pattern is observed even when the IoU threshold is lowered. This is not particularly surprising given that all training images share the same grid of bounding boxes. The model essentially memorizes the positions of the bounding boxes and understands that there should be no overlaps, resulting in consistent performance regardless of the IoU threshold.



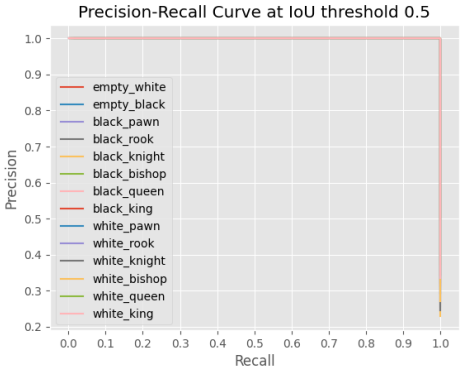
(a) Precision Confidence Curve



(b) Recall Confidence Curve



(c) F1 Confidence Curve

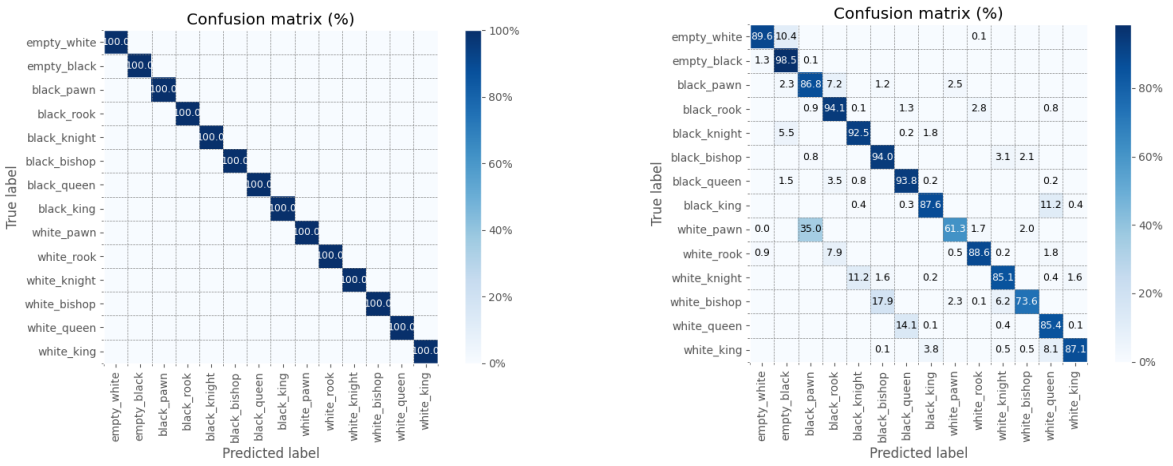


(d) Precision Recall Curve

Figure 4: Metrics obtained on the test set with a IoU threshold of 0.5

For the classification problem at hand, we can construct a confusion matrix at varying threshold levels. The matrices for two separate confidence levels are illustrated in Figure 5. We deliberately selected contrasting levels to demonstrate the proficiency of the model and to identify the thresholds at which errors eventually occur. The IoU threshold is consistently held at 0.5, but any level slightly above 0 would suffice. The matrix further reaffirms the model's exceptional performance when an appropriate confidence threshold is applied.

Examining the example with a low confidence threshold (0.005), we note that most misclassifications occur between pieces of different colors. A few interesting observations can be drawn from this analysis. For instance, an empty black tile is seldom confused with a white one (1.3%), but the converse isn't always true (10.4%). The model occasionally confuses the black king with the white queen (11.2%) and mistakes the white king for the white queen (8.1%).



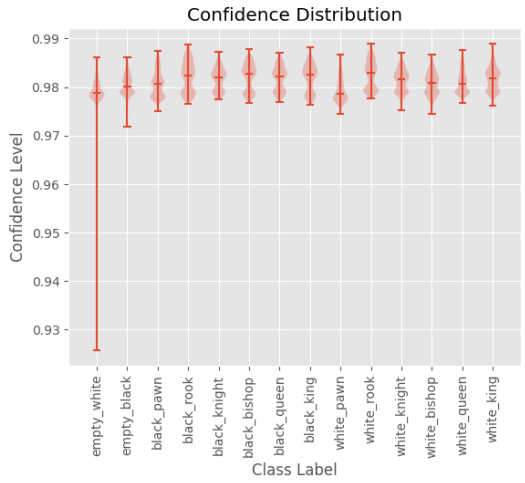
(a) Confidence Threshold: 0.75

(b) Confidence Threshold: 0.005

Figure 5: Confusion Matrix of the prediction on the test set at two different confidence thresholds, both with a IoU threshold of 0.5

Finally, considering our primary objective of predicting the correct Forsyth-Edwards Notation from an image, we can readily convert the top 64 predictions by confidence into a FEN string. Figure 6a illustrates the distribution of confidence scores across the entire test dataset, selecting the top 64 objects, demonstrated through a violin plot. Overall, it is evident that these objects display extremely high confidence scores, reinforcing the model's impressive performance.

In Figure 6b, a visual representation of the model's performance on a single example is provided. Here, we have overlaid the predicted bounding boxes, class labels, and confidence scores. Once again, no misclassification is seen, with high confidence across all tiles, underscoring the model's reliability.



(a) Distribution of the confidence across all samples of the top 64 predictions

(b) Visualization of the prediction of the objects with the top 64 confidence scores

Figure 6: Prediction of the objects with the top 64 confidence scores

The actual process of converting the prediction to a string presented some complexities, as we had to arrange the bounding boxes in relation to one another. A straightforward ordering based on the x and y coordinates proved insufficient, as even minor inaccuracies in the predictions could lead to discrepancies due to small fluctuations in these values. However, we managed to circumvent this problem by appropriate rounding of the values.

A final comparison of the Forsyth-Edwards Notation (FEN) strings between the ground truth and predicted data resulted in an accuracy of 100%. Given all the previous performance metrics we considered, this high level of accuracy is not surprising.

5 Conclusion & Outlook

Overall, this project has demonstrated the efficacy of object detection for the task at hand. We successfully built a pipeline that almost flawlessly predicts the correct Forsyth-Edwards Notation (FEN) string from a given screenshot.

When it comes to code adaptation, we encountered some challenges, particularly in data generation, which involves exact cropping and determination of bounding box coordinates — variables that can differ from one computer to another — and the extraction of personal cookies. Training largely followed the procedure outlined in the tutorial, with minor adjustments necessary in the creation of a custom yaml file and the modification of training parameters. The analysis was built almost entirely from scratch, thereby proving relatively time-consuming.

It's essential to note that the data used here represents an ideal scenario, with images already cropped to display only the chess board. In a real-world setting, we would need to first detect the board and appropriately resize the image or train the object detector directly from a variety of hand-annotated full screenshots. Using a smaller model could also be

beneficial to reduce inference time. Finally, to validate the model's effectiveness, it should be evaluated on feasible, and potentially even real, historical chess games. This would provide a more realistic test of the model's predictive capabilities in practical applications.

References

[1] Chess pieces object detection in 15 minutes. Blog post. https://www.instructables.com/Chess-Pieces-Object-Detection-in-15-Minutes/.

[2] Roboflow. Chess full. Roboflow Public Datasets. https://public.roboflow.com/object-detection/chess-full.

[3] Elucidation Sam. tensorflow_chessbot. https://github.com/Elucidation/tensorflow_chessbot/tree/master, 2016.

[4] Ultralytics. Yolov5: Train custom data. Online tutorial. https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/.

[5] Ultralytics. Yolov5: v5.0. https://github.com/ultralytics/yolov5, 2023.