Chessboard state extraction using the DEtection TRansformer

Bhatti Roben
roben.bhatti@studenti.unipd.it

Panagiotakis Konstantinos
konstantinos.panagiotakis@studenti.unipd.it

Sapkas Michail
michail.sapkas@studenti.unipd.it

Abstract

We explore the power of the Transformer architecture in the object detection task. Given an image, the model has to predict bounding boxes associated with a class. This novel and elegant approach, is using the best features of both CNNs and Transformers by omitting a lot of previously hand crafted features and achieving accuracies comparable to state of the art vision systems. We apply this approach on a chess dataset, with the goal of extracting the state of the game by detecting all pieces and the chessboard corners.

1. Introduction

Transformers have revolutionized NLP models by introducing the attention mechanism [1]. Now, a lot of work is being re-directed on the task of using this powerful mechanism on vision tasks, such as object recognition and panoptic segmentation. In this work, we apply the Detection Transformer, a model that uses a pretrained CNN as backbone to create feature maps which then are fed into the Transformer, involving the attention mechanism. Attention will learn to capture meaningful correlations between pixels in order to output learnt representations of objects in the image. We apply this technique to a chessboard dataset with the goal of correctly identifying the chessboards corners and the chess pieces, then, assign them to a lattice constructed using the corners and finally, output the positions of all the chess pieces as the state of the chess game at a particular moment. We benchmark the results with a YOLOv8 model also pretrained on this task. The code can be found in [2].

2. Related Work

2.1. DETR paper

The DETR model was proposed in End-to-End Object Detection with Transformers by Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov and Sergey Zagoruyko [3]. DETR consists of a convolutional backbone followed by an encoder-decoder Transformer which can be trained end-to-end for object detection. It greatly simplifies a lot of the complexity of models like Faster-R-CNN and Mask-R-CNN, which use things like region proposals, non-maximum suppression procedure and anchor generation. DETR eliminates the need for these components, presenting a unified framework that directly predicts object bounding boxes and class labels in a single forward pass.

2.2. Applying Vision in Chess

We were inspired by the application of real-life-chess-vision[4] on github. The model takes images or video frames of a chessboard as input. This can be from a camera capturing a physical chessboard or screenshots from a digital chessboard. Then a vision model, in this case YOLOv8[5], is trained to detect and locate chess pieces within the images and the corners of the chessboard. This, first involves recognizing the different types of pieces (pawn, knight, bishop, rook, queen, and king) and determining their positions on the board. Then, by transforming the depth field parallelogram defined by the chessboard corners to a square grid, we can assign square positions to the pieces. The final output, is a representation of the chessboard state, typically in the form of a string.

This approach allows for automated and real-time tracking of the chessboard state, enabling applications such as computer vision-based chess-playing robots, virtual chess analysis tools, or assisting players in recording games.

3. Dataset

The DETR foundation model, was trained with the COCO 2017[6] (Common Objects in Context) dataset, which means that using the code provided by the developing team to fine-tune it, we will need to provide Datasets that are annotated and structured exactly as the COCO format.

3.1. The COCO Dataset Format

The COCO dataset is a widely used benchmark in computer vision and object recognition research.

The images are stored in JPEG format, while annotations are provided in JSON format. Each annotation contains details about the category, bounding box coordinates, and segmentation mask.

Of our interest, are the bounding box coordinates and the associated class prediction, which are given in a vector format containing first the x, y coordinates of the upper left corner of the box, accompanied by the $width$ and the $height$ of the box:

$$bbox : [x, y, width, height]$$

3.2. Data Acquisition

In order to acquire the necessary datasets we used the online platform for computer vision, provided by Roboflow[7]. Roboflow allows users to upload custom datasets and draw annotations. It hosts free public computer vision datasets in many popular formats, including COCO JSON, which is the one we are interested in.

3.3. Chess pieces Dataset

The pieces dataset [8] contains photos 416×416 captured from a constant angle, a tripod to the left of the board (fig. 1). The bounding boxes of all pieces are annotated as follows: white-king, white-queen, white-bishop, white-knight, white-rook, white-pawn, black-king, black-queen, black-bishop, black-knight, black-rook, black-pawn. There are 2894 labels across 292 images.

The training set of the chess pieces consists of 606 images of variously populated chessboards which have been augmented 3 times from the original 292 images by horizontal flip, crop, shear and hue, saturation, brightness and exposure variations.

3.4. Chessboard corners Dataset

The corner detection dataset [9] contains 768 total images, an example is shown on fig.2. The images have been uniformly resized to dimensions of 640×640 pixels and data augmentation techniques have been applied to augment the dataset. Such as Horizontal and vertical flips, along with 90° rotations in both clockwise and counter-clockwise directions, and upside-down flips have been employed to introduce variations and promote model generalization. A touch of gray scale has been added to approximately 15% of the images, imparting an additional dimension to the dataset. Hue adjustment, spanning a range between -90° and +90°, has been implemented to infuse color diversity. The introduction of noise, limited to 2% of pixels, serves to simulate real-world scenarios. Furthermore, a cutout strategy has been adopted, involving the removal of three rectangular regions, each covering 10% of the image size, fostering adaptability in the face of occlusions and irregularities.

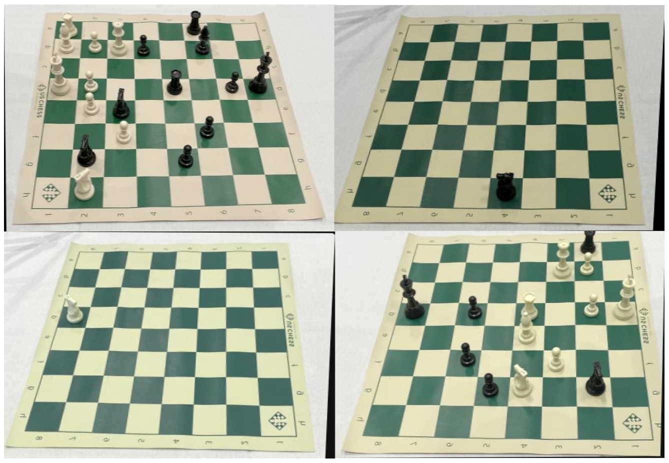
Figure 1. Chess pieces dataset

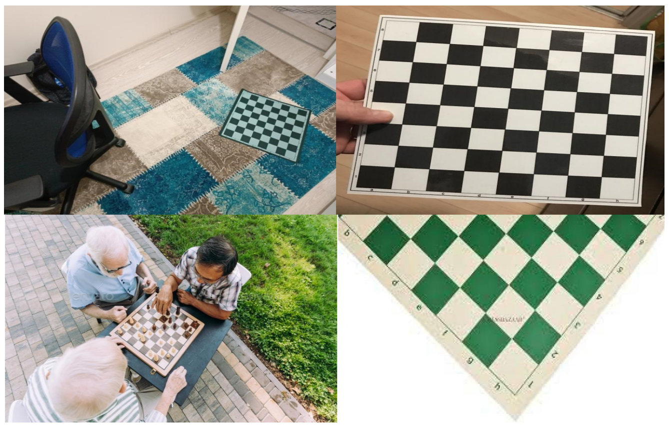
Figure 2. Chessboard corners dataset

4. Methods

As mentioned earlier, our goal is to get the positions of the chess pieces from the image of a real world chessboard using the DETR and converting it to a Forsyth–Edwards Notation (FEN) annotated board. In the original commit, this task was accomplished with a foundational YOLOv8 model.

4.1. DETR model

As for the actual model code, we used the implementation provided by Hugginface[3].

The DETR model relies on two key ingredients: a robust CNN backbone, which in our case is the ResNet-50, and a standard Transformer. ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one Max-

Pool layer, and one average pool layer). Residual neural networks are a type of ANNs that forms networks by stacking residual blocks. The number of the ResNet-50 output feature maps get reduced, typically from $2048 \times \frac{H}{32} \times \frac{W}{32}$ to a lower number $d \times \frac{H}{32} \times \frac{W}{32}$ using a 1×1 kernel convolution, and then, the dimensions get flattened to a sequence of $d \times h \cdot w$ vectors. Having effectively converted the feature maps into a sequence of vectors, positional encoding is added (typically sinusoidal). Then, the encoder processes the input image and the decoder generates object queries and refines their positions.The model simultaneously considers all object queries and their interactions, enabling a holistic understanding of the scene.

DETR utilizes a bipartite matching loss to establish associations between predicted and ground truth objects, ensuring accurate localization and classification. Specifically, the predicted classes and bounding boxes for each of the $N = 100$ object queries are compared to the ground truth annotations, padded to match the length N. In cases where an image contains only a few objects, the remaining annotations have a ”no object” class and ”no bounding box.” The Hungarian matching algorithm is employed to find an optimal one-to-one mapping between the N queries and annotations. Subsequently, standard cross-entropy is used for the classes, while a linear combination of L1 and generalized IoU loss is employed for the bounding boxes to optimize the model parameters.

4.2. Fine Tuning the DETR

In order for the model to specialize on the specific chess task, we need to further train the DETR foundation on the two custom datasets.

We fine-tune two separate models, each performing a detection task but for different targets. The first model detects and classifies the chess pieces, whereas the second model detects the corners of the chessboard. We followed this approach mainly for the reason that, even though Roboflow allows the merging of datasets into a commonly annotated dataset, merged datasets exhibited images annotated exclusively with either chess pieces or corners, failing to fulfill the dual objective of simultaneous detection. Despite the potential inconvenience of managing two task-specific models, this pragmatic approach was adopted to maximize accuracy.

4.3. Creating Chessboard Grid using the Detected Chessboard Corners

Once we have detected the chess pieces, we need to assign them into positions on the chessboard squares (eg. C4, B8 etc). This is were the detection of the chessboard corners comes into play.

The difficulty lies on the fact that most of the time the chessboard is being captured at an angle, resulting in a stretched representation of the grid, were distances are not Cartesian. We have to find a linear transformation that will map coordinates from a parallelogram, that the detected corners define, to a square. The transformation is defined by four source points (the detected corners) and their corresponding destination points (corners of a square defined by the parallelograms max width and max height).

Once we have acquired the transformation matrix, we can apply the transform to all the detected boxes coordinates, effectively mapping them to a flat surface. A rescaling of the bounding boxes is necessary in order for the upcoming assignment to be successful.

For visualization purposes, we can also crop and transform the chessboard itself. The resulting image, represents the original image with a changed perspective to that of ”bird’s eye view”, that is from on top.

At this point we create an auxiliary 8x8 chessboard grid, which contains 64 bounding boxes. This boxes represent the chessboard squares and fit exactly on our chessboard.

The assignment of a piece to a square, is performed by calculating the IoU between the bounding box of the piece and all the squares. For each detection, we assign the square position with the largest IoU.

Finally, by having assigned every detection to a square, we output the state of the chessboard in FEN to be visualized neatly by dedicated chess software. The whole process is also shown in fig.3

Table 1. Comparison of DETR and YOLO on corners and pieces datasets. AP and AR with IoU=0.5

Dataset	DETR		YOLO	
	AP	AR	AP	AR
Pieces	0.94	0.78	0.66	0.99
Corners	0.17	0.13	0.071	0.87

5. Experiments

Since the original idea was implemented using the YOLOv8 model, we will use it to benchmark the DETRs efficiency. The tasks can be separated into the two distinct detection targets, chess pieces and corners.

The experimental results where acquired using the test dataset of the chess pieces. The dataset consisted of 29 images.

We use two very standard metrics for benchmarking, Average Precision and Average Recall. Then we also compute the confusion matrix for both models.

5.1. Metrics

Average Precision (AP) is an evaluation metric commonly used in the context of object detection.

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It answers the

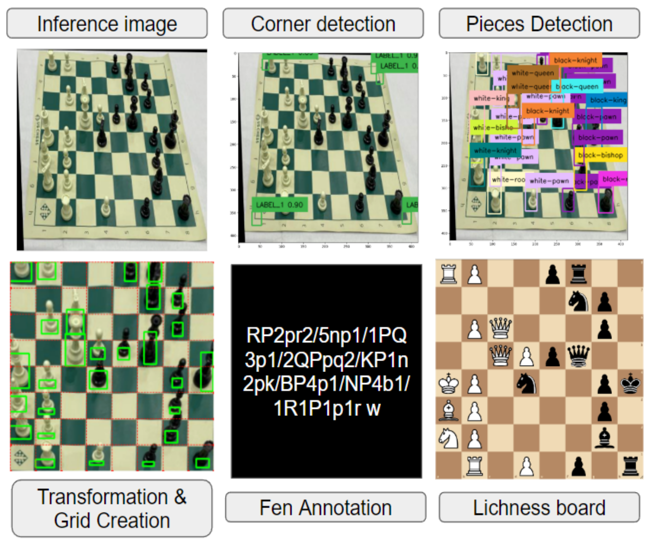
Inference image
Corner detection
Pieces Detection
Transformation & Grid Creation
Fen Annotation
RP2pr2/5np1/1PQ3p1/2QPpq2/KP1n2pk/BP4p1/NP4b1/1R1P1p1r w
Lichness board

Figure 3. Chessboard state to computer model. As real chess board state is taken as input, the image should have visible corners. First inference is done by the corner detection DETR model followed by the DETR pieces detection model which labels each piece. Then, we transform the image so that the chess board's boxes are orthogonal from one another and a grid is superimposed to the image. Each labelled box is then assigned to a square by searching for the highest IoU between the box and the grid. Finally, the board is converted into a FEN format that can be read by any dedicated chess playing program, to understand the state of the game.

question, "Of all the instances predicted as positive, how many were actually positive?"

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

In summary, Average Precision provides a single scalar value that summarizes the Precision-Recall performance of an object detection model across different confidence thresholds.

Recall, also known as sensitivity or true positive rate, is a measure of how well a model captures all relevant instances of a particular class. It is defined as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Average Recall extends this concept to multiple classes by calculating the average recall across all classes. The formula for Average Recall is:

$$\text{Average Recall} = \frac{1}{N} \sum_{i=1}^{N=Classes} \text{Recall}_i$$

In the context of vision models and object detection tasks, a high Average Recall means that the model is effective at detecting objects across various classes, ensuring

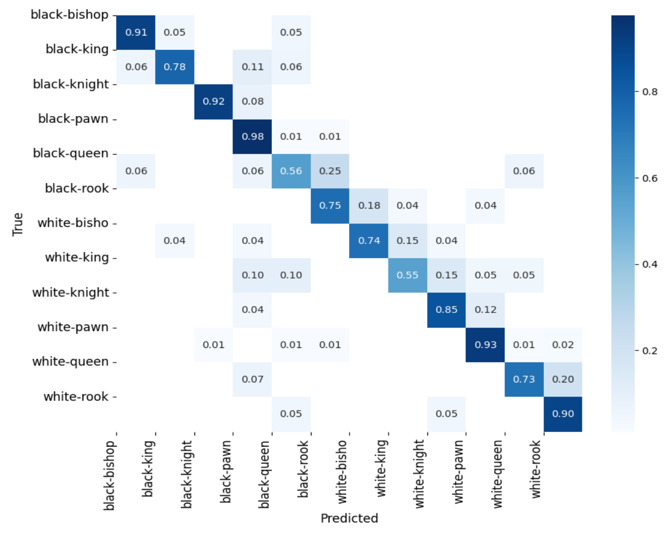

Figure 4. DETR Confusion Matrix

that it doesn't miss many instances of different objects in the images.

Average Precision Calculation:

$$\text{AP} = \frac{1}{n} \sum_{r \in \text{recall}} \text{precision}(r)$$

Average Precision provides a comprehensive summary of a model's performance across different confidence levels, with a higher value indicating better overall precision-recall trade-off.

A confusion matrix is a table used in classification to assess the performance of a machine learning model. It summarizes the model's predictions compared to the actual outcomes across different classes.

Confusion matrices (figure 4 and 5) provide a detailed view of a model's performance, enabling a deeper understanding of its strengths and weaknesses in differentiating between classes.

To compute the DETR confusion matrix was tricky because sometimes it detects the same piece twice, specifically in scenarios where the piece is tall in dimensions (eg. Kings, Queens, Bishops) and another piece is directly above.

In order to handle false positives, that is a piece is detected but its not classified correctly, we employ a trick in which a random (but not of the same class) piece gets added as ground truth and directly corresponds with the duplicate detection.

6. Conclusions

From the acquired Avg. Precision and Avg. Recall (table 1) we derive some key observations:

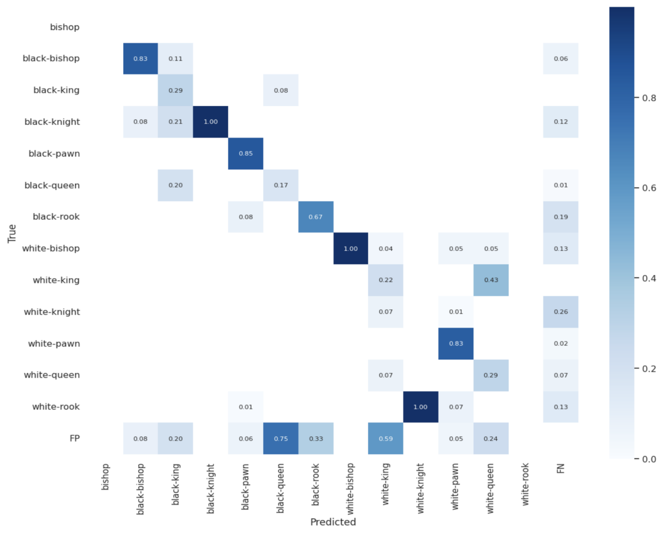

Figure 5. YOLOv8 Confusion Matrix

DETR performs excellent on piece detection with respect to YOLOv8. Corner detection appears to be hard for both models. We suspect that the main problem is the small scale of the corners features since, in general, vision models and especially DETR operate better above a feature scale threshold. The fact that DETR scores high in Avg. Precision and low in Avg. Recall indicates that the model is good at correctly identifying positive instances when it predicts them (few false positives), but it is not effective at capturing all the positive instances present in the dataset (high false negatives). As mentioned above, in some circumstances, DETR created duplicate detections. We believe that this problem should be addressable with further training on bigger datasets. Precision and Recall are often trade-offs. As you increase one, the other may decrease. We clearly see this on our results. Its important to note that the performance of these models heavily depends on the quality and diversity of the training data. Adequate and representative datasets are essential for training models to generalize well to different chessboard scenarios. To conclude, given the limitations of our datasets and computational power we think that the DETR model, given in its vanilla form, performs in a manner at least comparable to YOLOv8, which is not a insignificant feat. Future work could include adding procedures that make the model more robust on image transformation and square assignment (such as neglecting detected pieces that are not in play), possible real time detection using video feed or finally, API connection with chess gaming software in which the user could play chess in real life and the software responds.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[2] Sapkas M. Panagiotakis K, Bhatti R. Github repo with the code. `https://github.com/kostas-panagiotakis/Vision/tree/main`, 2023.

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *CoRR*, abs/2005.12872, 2020.

[4] Ph.D. Shai Nisan. real-life-chess-vision. `https://github.com/shainisan/real-life-chess-vision`, 2023.

[5] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLO, January 2023.

[6] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[7] Nelson J. Solawetz J. et. al. Dwyer, B. Solawetz, j., et. al. roboflow (version 1.0) [software]. `https://roboflow.com`, 2022.

[8] 1. chess pieces dataset. `https://public.roboflow.com/object-detection/chess-full`, dec 2023.

[9] 1. chess table corners dataset. `https://universe.roboflow.com/1-zdtjh/chess-table-corners`, dec 2023. visited on 2024-01-22.