Chessboard state extraction using the DEtection TRansformer

Roben Bhatti - 2091187
Konstantinos Panagiotakis - 2081260
Michail Sapkas - 2072109

Our Goal: Object Detection

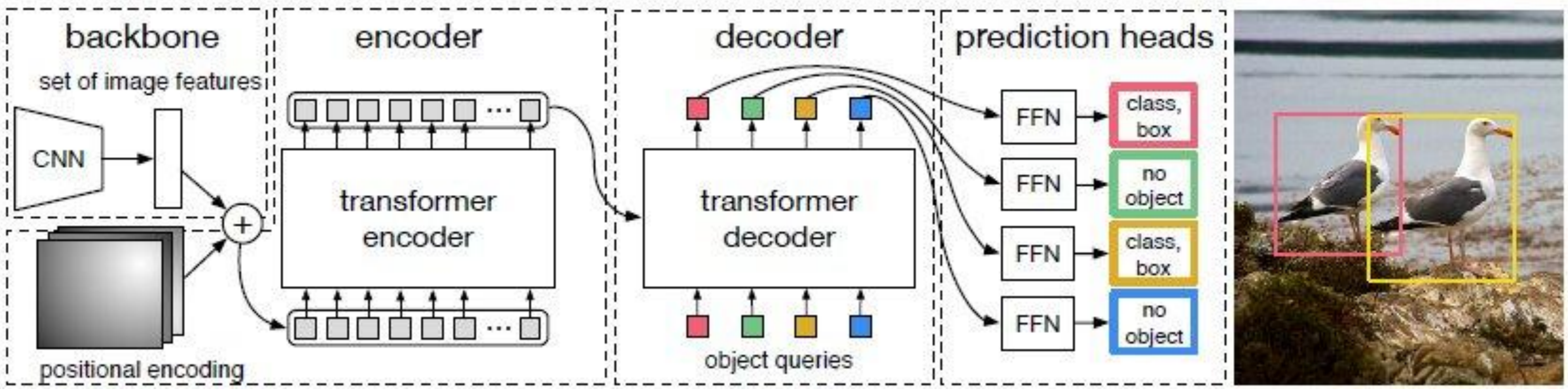
- Take a **real life chess** board game **state** and **express** it to a Forsyth–Edwards Notation, (**FEN**) annotation

- **Predict** the edge coordinates of bounding boxes, (**bboxes**) containing the objects in an image
    - Chess Pieces
    - Chess board corners

- Predict the **class** that each object belongs to

The DETR Model

- A pre-trained CNN layer (**Backbone**)
- An added **Conv2D layer** (to prepare **feature maps** for the transformer)
- A standard **Transformer Encoder – Decoder Layer**
- Two shared **Feed Forward Networks** for predictions (class, bbox)

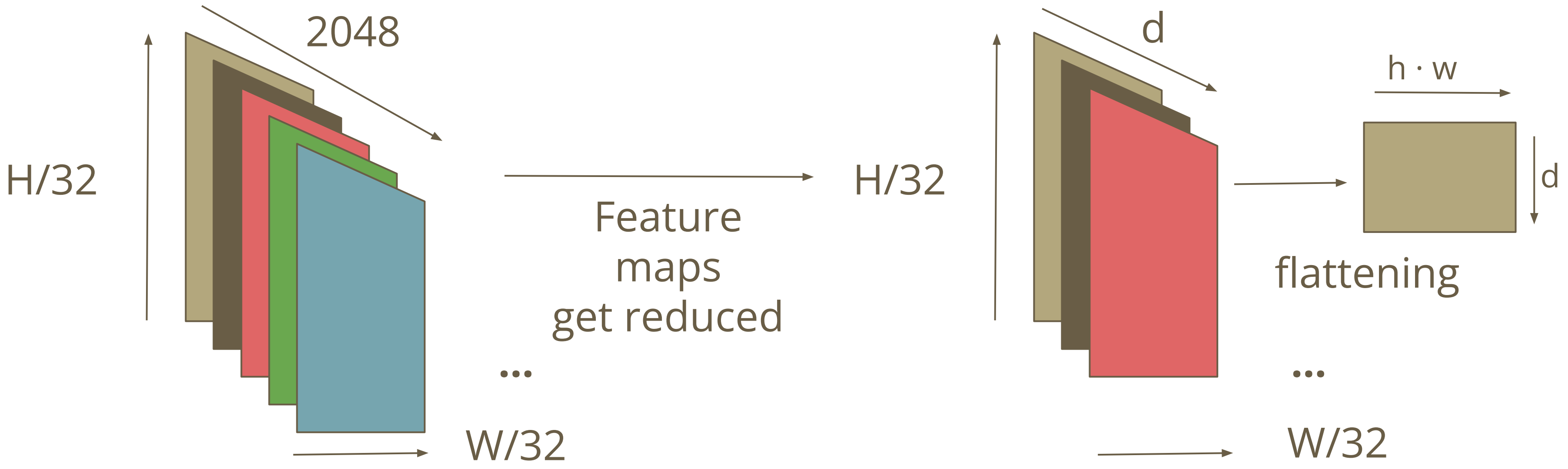


Convolutional Layers

- **ResNet-50** was used as the backbone of the model
    - ***ResNet-50*** *is a* ***50-layer CNN*** *(48 convolutional layers, one Max-2 Pool layer, and one average pool layer)*
- **Remove** the **Fully Connected Layers** of the ResNet50
- Connect the **Backbone** output (2048 x H x W) to another **CNN Layer**
- Serves as a **Feature Map reduction** by using a 1x1 kernel
- Reduces the Feature Maps from 2048 to 256, appropriate for transformer input

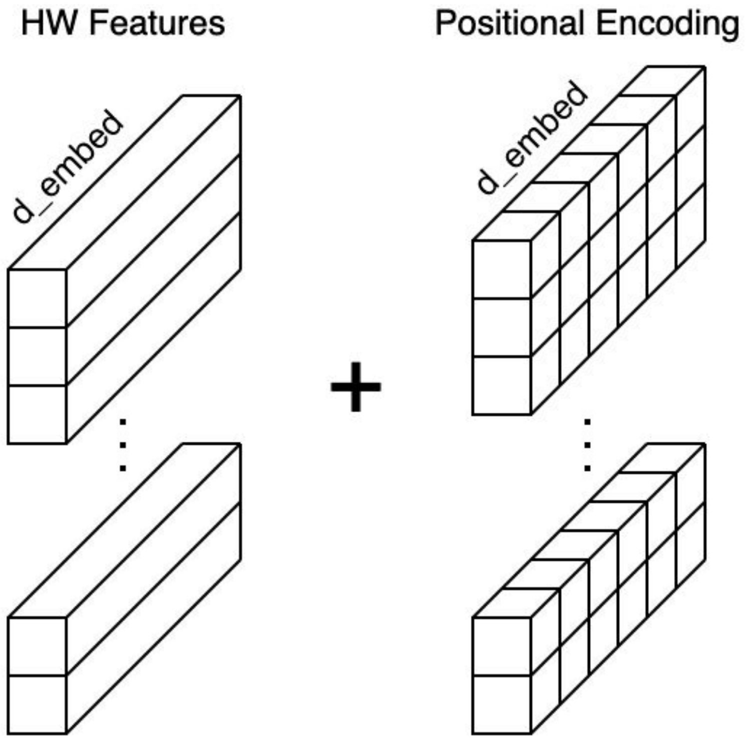
The second convolutional layer

- The number of the ResNet-50 output feature maps get **reduced**
  - Typically from **2048 × H/32 × W/32** to a lower number **d × H/32 × W/32** using a 1 × 1 kernel convolution, and then, the dimensions get flattened to a sequence of d × h · w vectors



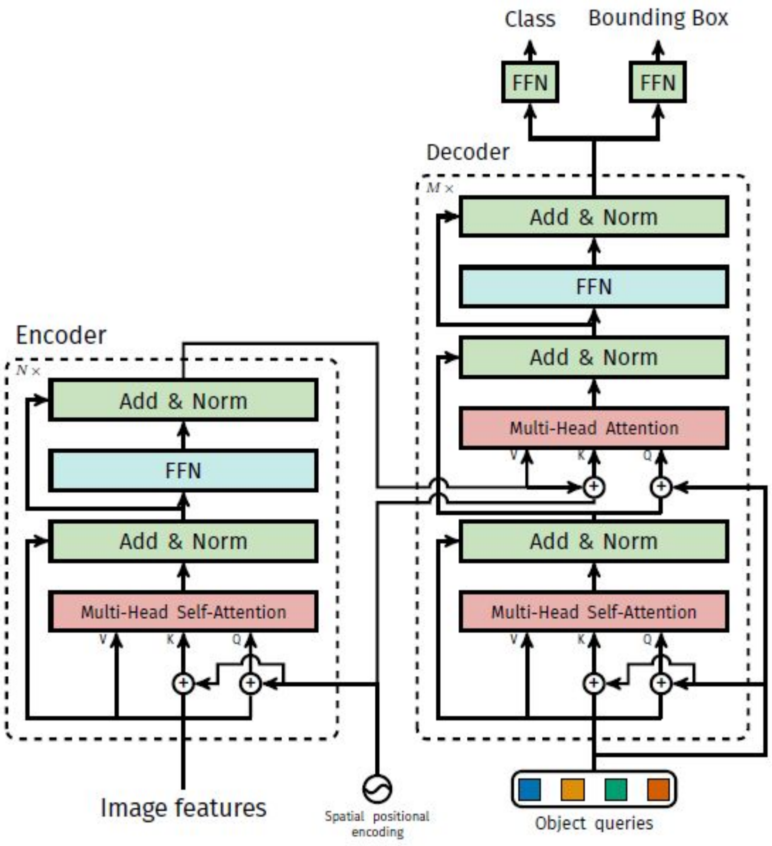
Visual Representation of positional encoding

- Positional encoding enables the model to effectively **capture spatial relationships between objects in images**
- Sum **fixed sine** functions to the embeddings
- added also to the self attention in the decoder



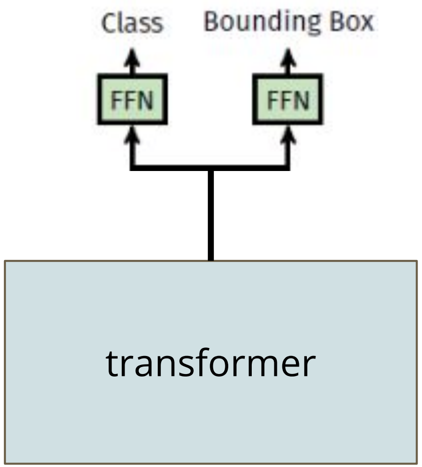
The Transformer

- The transformer is composed of the **encoder** and **decoder**
  - The encoder processes the input image
  - the decoder generates object queries and refines their positions
- Image **features** are **inputted** to the transformer encoder. The **decoder** takes **queries**, **positional encodings** and encoder memory to produce final predictions
- **Object queries** are learned vectors that the **transformer decoder** use to **output**
- They represent the **objects** in our **image** and the number of queries directly affects the number of objects the model can recognize.



Predictions

- Two **FFNs** share the same Transformer outputs (object queries) as inputs
- One FFN **classifies** the **objects**
- The other FFN **predicts** the **boundaries** of the **boxes**, the final prediction
- The "not a class" is a valid class object"

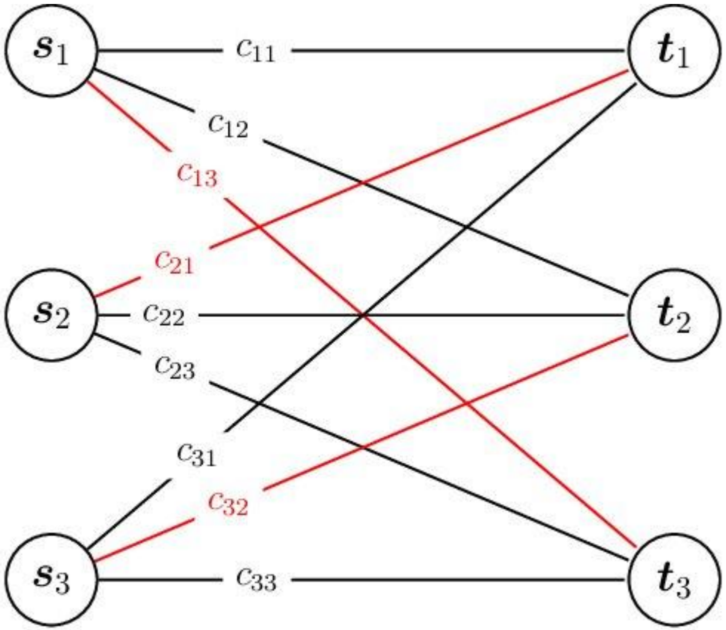


Bbox coordinates $b_i = [x, y]^4$

Bbox class $y_i = (c_i, b_i)$

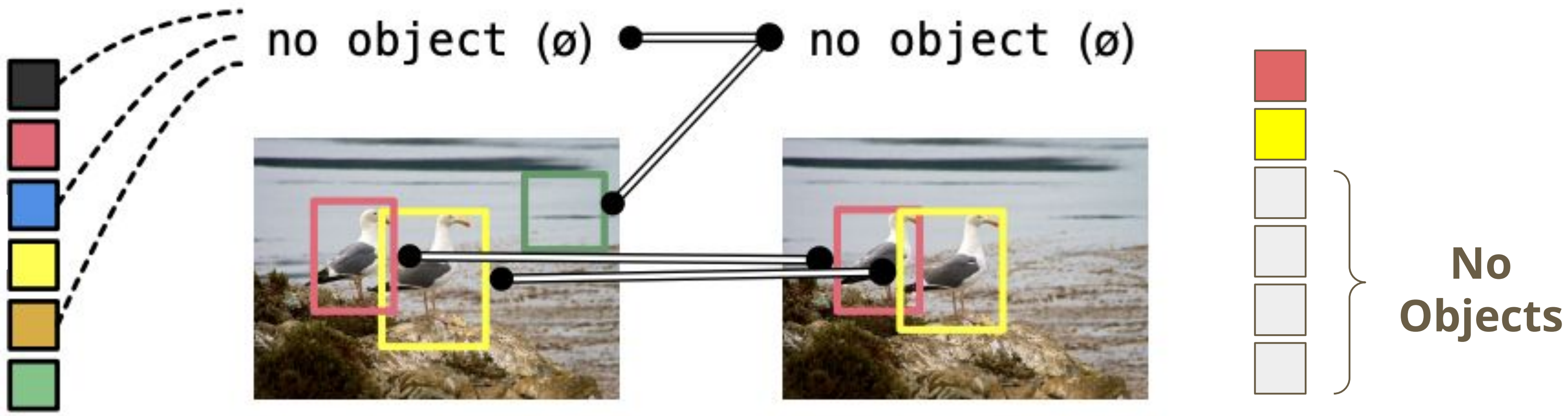
Bipartite Matching (Hungarian Algorithm)

- "Given a bipartite graph, a matching is a subset of the edges for which every vertex belongs to exactly one of the edges."
- Finds the correct permutation of our N predictions to **match** the **ground truth bounding boxes and class**
- Very useful because it **avoids** matching the same ground truth with more than one predictions



Bipartite Matching

Goal - Find a permutation of the set of prediction, σ, that minimizes the loss with the ground truth



Set of predictions **Bipartite Matching** **Ground Truth**

Matching Predictions to Ground Truth

- We need a way to **match** the **class** and **boundary box predictions** of the model to the ground truth (train dataset)
- This is done with a **bipartite matching** algorithm (**Hungarian Algorithm**)
- **The Hungarian Algorithm** needs a **cost matrix** which is going to be computed by our Loss terms
- L_{match} = pair-wise matching cost between ground truth y_i and a prediction with index σ(i).

$$\hat{\sigma} = \underset{\sigma \in \mathfrak{S}_N}{\arg\min} \sum_{i}^{N} \mathcal{L}_{\mathrm{match}}\left(y_i, \hat{y}_{\sigma(i)}\right)$$

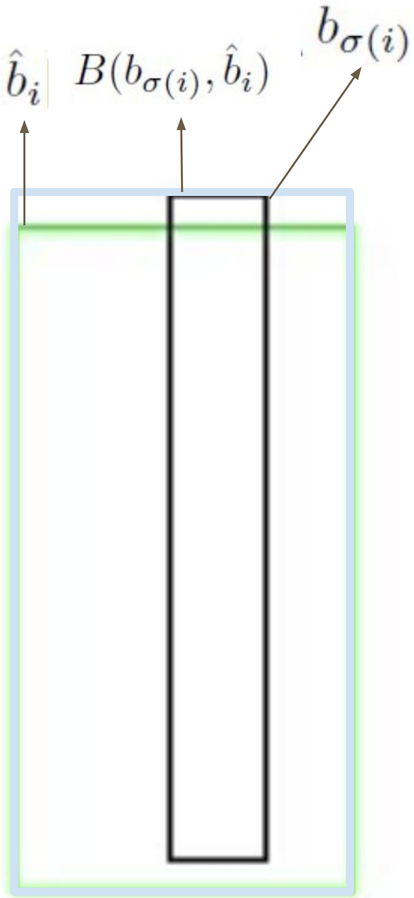
Cost for the Objective Function

$$\mathcal{L}_{\text{match}} = \underbrace{-\mathbb{1}_{\{c_i\neq\varnothing\}}}_{\substack{\text{= }\textbf{-1}\text{ when}\\ \text{class is not}\\ \text{No object}}}\underbrace{\hat{p}_{\sigma(i)}(c_i)}_{\substack{\textbf{Prob}\text{ of}\\ \text{A }\textbf{specific}\\ \textbf{Class}\text{ for}\\ \text{current}\\ \text{permutation}}} + \mathbb{1}_{\{c_i\neq\varnothing\}}\underbrace{\mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)})}_{\substack{\textbf{Loss}\text{ b/w}\\ \textbf{Ground truth \&}\\ \textbf{predicted bbox}\text{ for}\\ \text{current permutation}}}$$

$$\mathcal{L}_{\text{box}}(b_{\sigma(i)}, \hat{b}_i) = \underbrace{\lambda_{\text{iou}}\mathcal{L}_{\text{iou}}(b_{\sigma(i)}, \hat{b}_i)}_{\substack{\text{IoU }\textbf{Loss}\text{ b/w }\textbf{g.t and}\\ \textbf{predicted box}\text{ for}\\ \text{Current permutation}}} + \underbrace{\lambda_{\text{L1}}\|b_{\sigma(i)} - \hat{b}_i\|_1}_{\substack{\textbf{Distance}\text{ b/w g.t and}\\ \text{predicted boxes for}\\ \text{Current permutation}}}$$

Generalized Intersection over Union (GIoU)

$$\mathcal{L}_{\text{iou}}(b_{\sigma(i)}, \hat{b}_i) = 1 - \left(\frac{|b_{\sigma(i)} \cap \hat{b}_i|}{|b_{\sigma(i)} \cup \hat{b}_i|} - \frac{|B(b_{\sigma(i)}, \hat{b}_i) \setminus b_{\sigma(i)} \cup \hat{b}_i|}{|B(b_{\sigma(i)}, \hat{b}_i)|} \right)$$



- Two terms define the G-IoU:
  - The **ratio** between the **intersection** over the **union** of the predicted & ground truth boxes - **The IoU**
  - The **ratio** between:
    - the largest box containing the predicted and ground truth boxes, **B**, divided by the **union** of the predicted & ground truth boxes
    - **B**

Hungarian Loss

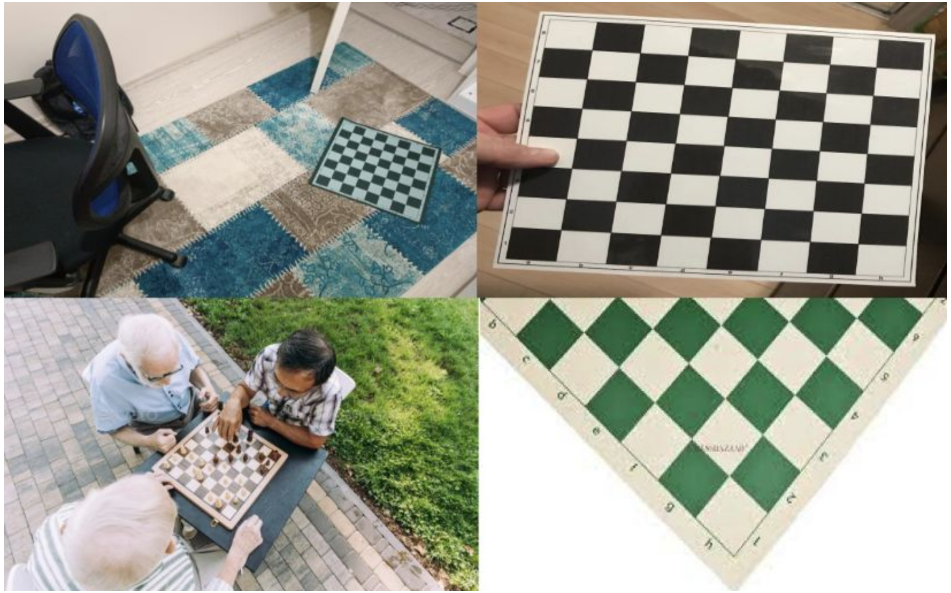
- Once the right permutation that minimizes the match loss has been found:

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

- Minus the log of the **probability** that the **classification** is the **same** with the ground truth
- When $c_i = \varnothing$, the log-probability term gets down-weighted by a factor 10

Dataset - Chessboard Pieces & Corners

Chess pieces ds

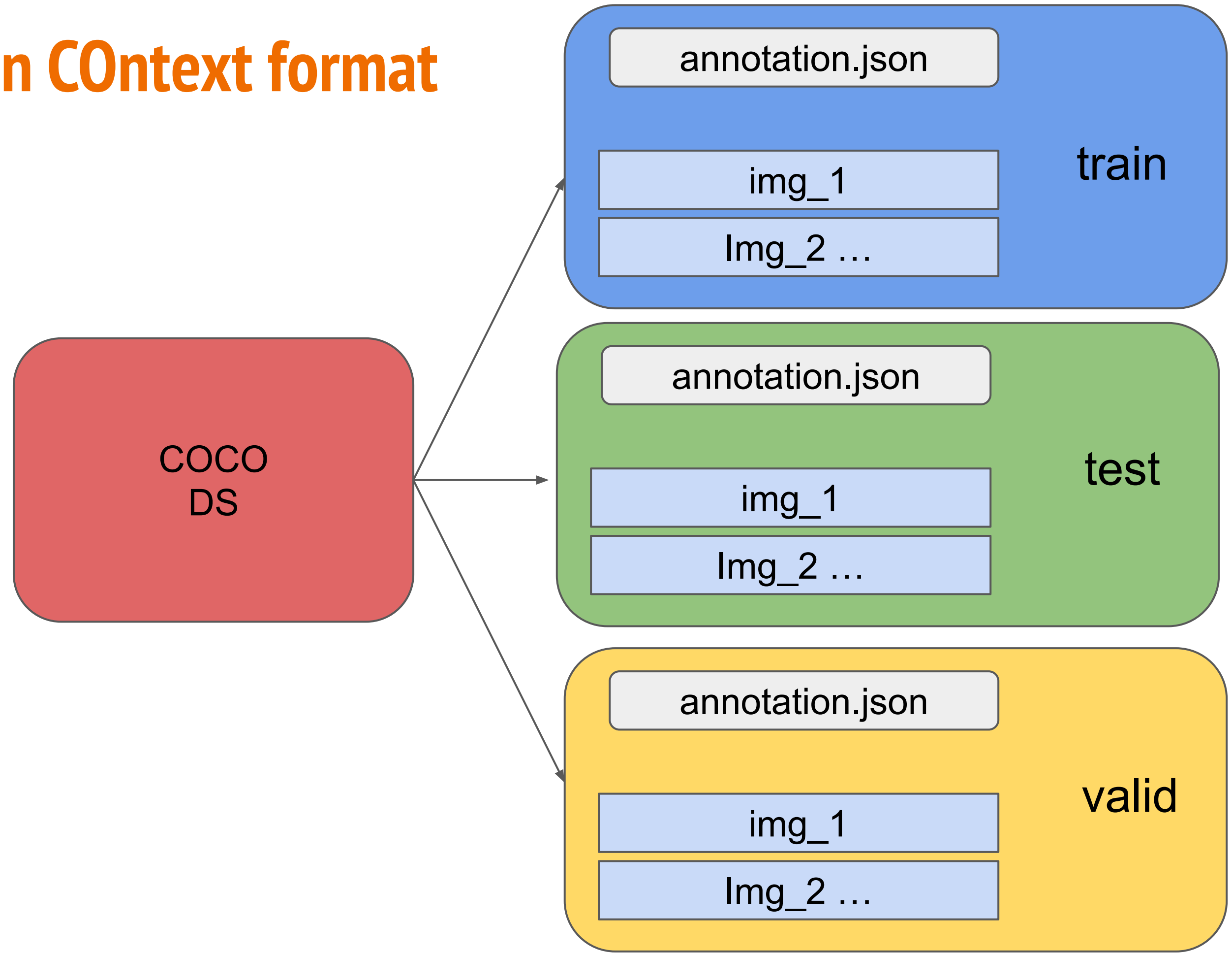


Chess corners ds

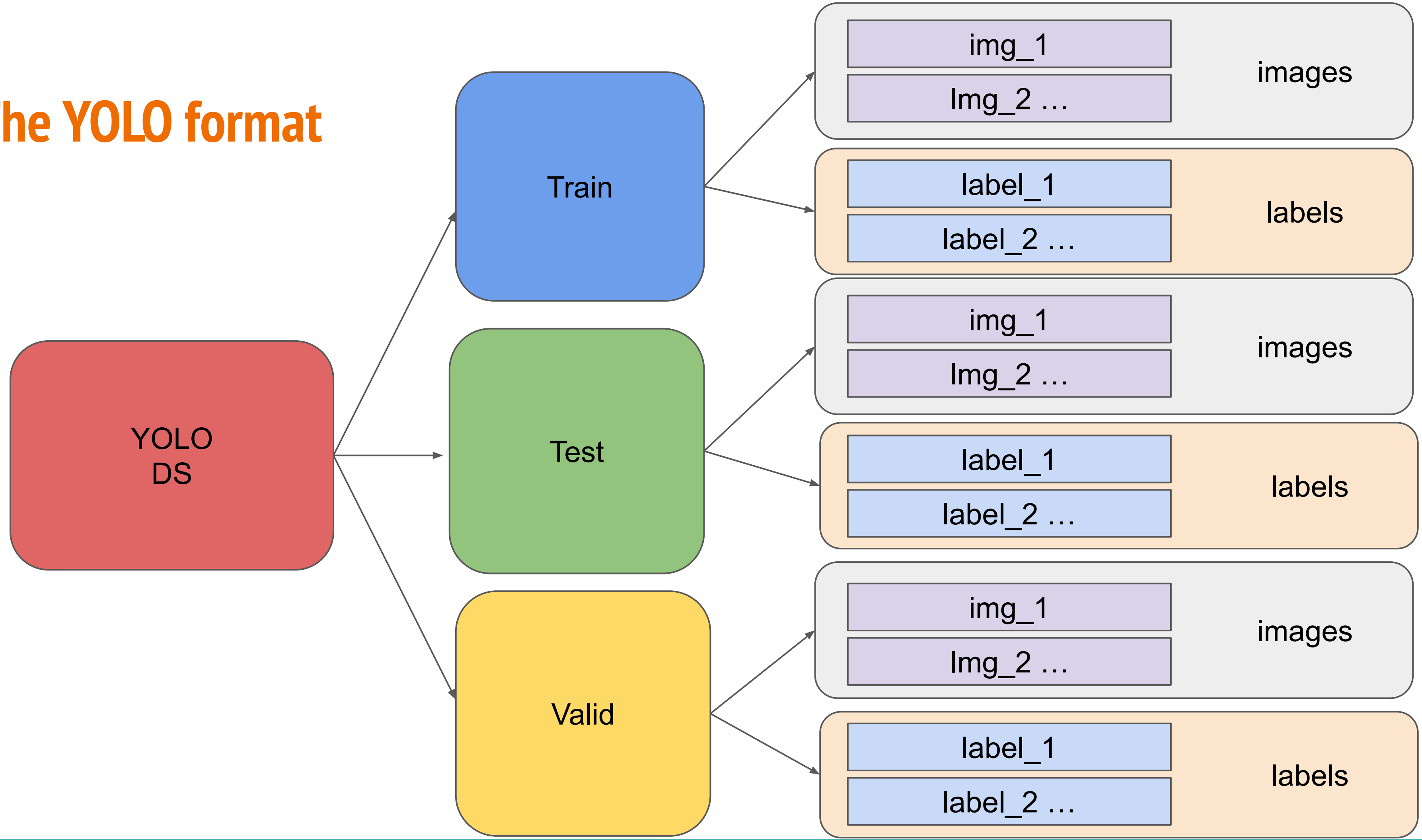
Deep dive in the DS

- Both DS were collected from Roboflow
- Chess Corners:
  - **768 images** of **640 x 640** - captured at the different angles
  - **2 classes**, corner, no corner
  - Training - 603 images, augmented with flips, hue adjustment, noise
  - Test 62 & Validation 103
- Chess Pieces:
  - **292 images** of **416 x 416** - captured at the same angle
  - **13 classes**, one for each chess piece + 1 for “bishop” and + 1 for “pieces”
  - **Format** label: color - piece
  - Training - 606 images, augmented 3 times by original 292 images with multiple transformations
  - Test 29 & Validation 58

The Common Objects in COntext format



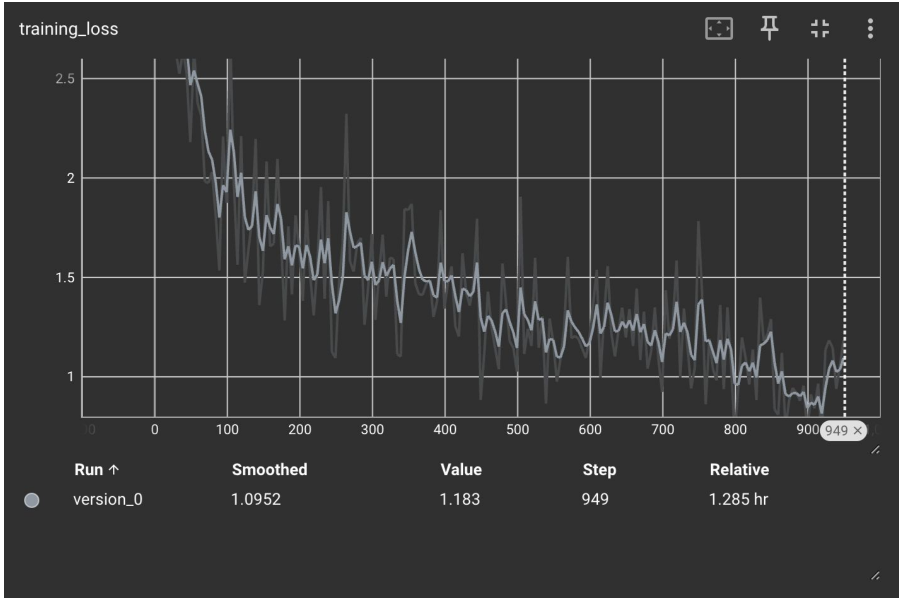
The YOLO format

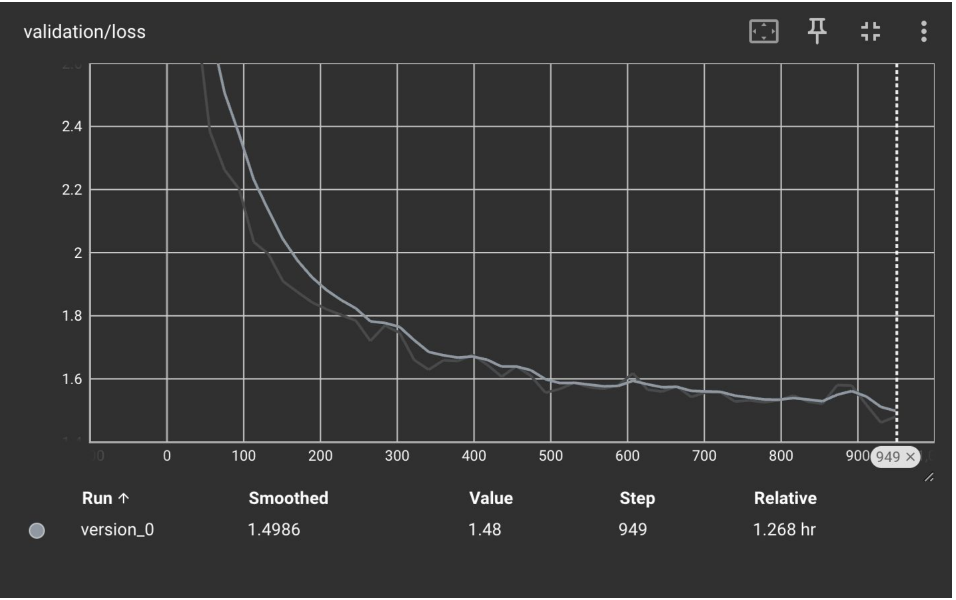


DETR Fine Tuning

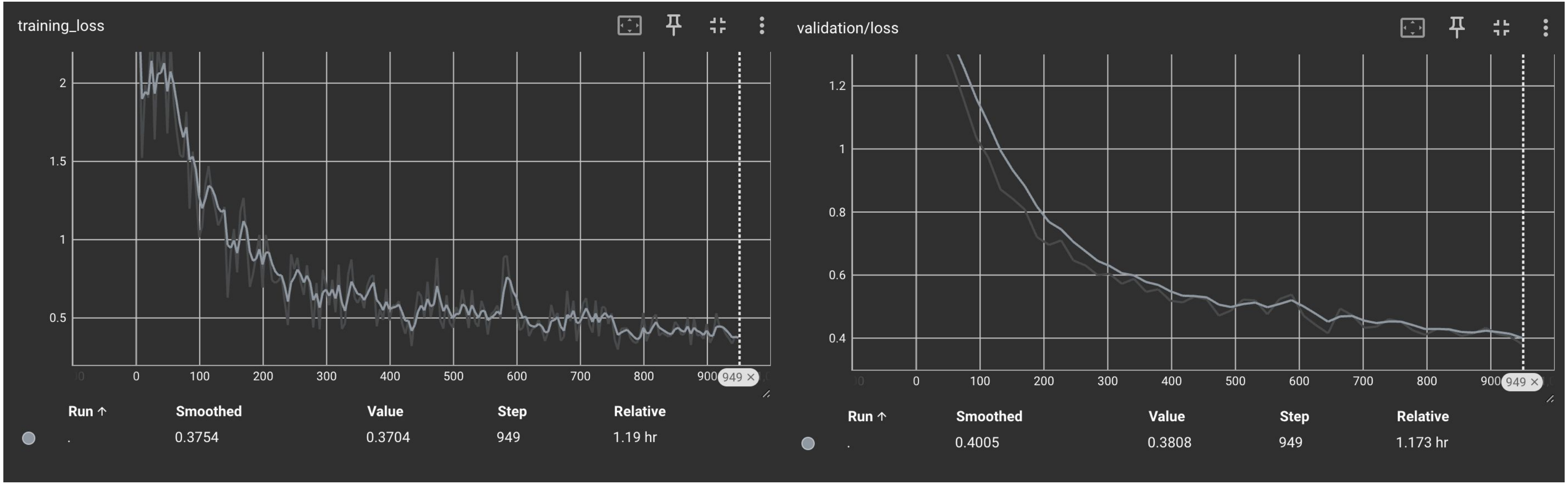
- We used **DetrForObjectDetection** model from **Huggingface** pre-trained on the COCO dataset
- **Two separate models** are fine-tuned for different detection tasks:
  - First model detects and classifies chess **pieces**
  - Second model detects **corners** of the chessboard
- Relevant hyperparameters of fine tuned DETR model are the same as the source model
- We trained on 30 number of epochs

DETR Corner Losses





DETR Pieces Losses



Inference

- The **DETR** fine tuned model is used to obtain:
  - The boxes for the corners of the chessboard
  - The labels and the boxes for each of the chess pieces on the board
- Chessboard grid creation:
  - **Detected** chess **pieces** need to be assigned to positions on the chessboard squares.
  - Detection of chessboard **corners** is crucial for this task
- **Challenge**: Captured chessboard is often at an **angle**, causing a stretched perspective of the chessboard square grid
- **Solution**: Apply a **Linear transformation** from a **parallelogram** defined by four **source points** (using the detected corners) to a **rectangle**

Corner detection

Pieces Detection

Piece assignment process

- Transformation process:
  - **Acquire transformation matrix using cv2**
  - **Apply transformation to all detected box coordinates**
  - **Rescale bounding boxes for successful assignment**
- Visualization: **Crop** and **transform** chessboard for "bird's eye view" perspective
- Auxiliary 8x8 chessboard **grid**, contains **64** bounding **boxes**, that correspond exactly on the squares of the chessboard
- **Piece assignment**: Calculate Intersection over Union (IoU) between piece bounding box and all squares in the grid
- **Assign** each **detection** to the square with the **highest IoU**
- **Output**: State of the chessboard in **FEN format** for visualization in dedicated chess software

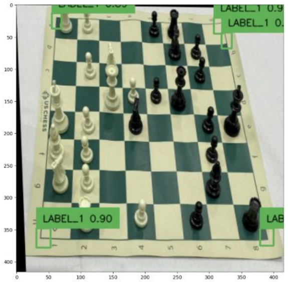
Inference image

Transformation & Assignment

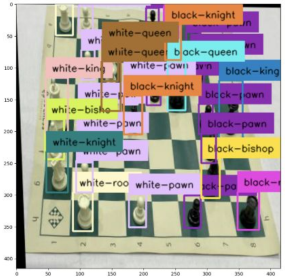
Visualization of piece assignment

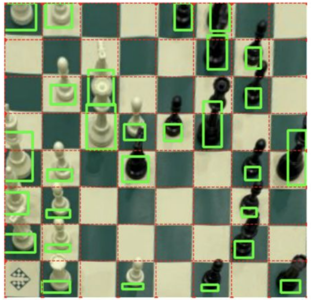
Inference image

Corner detection



Pieces Detection

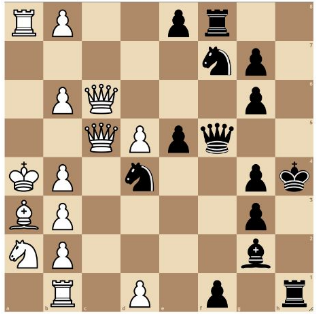




Transformation & Assignment

RP2pr2/5np1/1PQ
3p1/2QPpq2/KP1n
2pk/BP4p1/NP4b1/
1R1P1p1r w

Fen Annotation

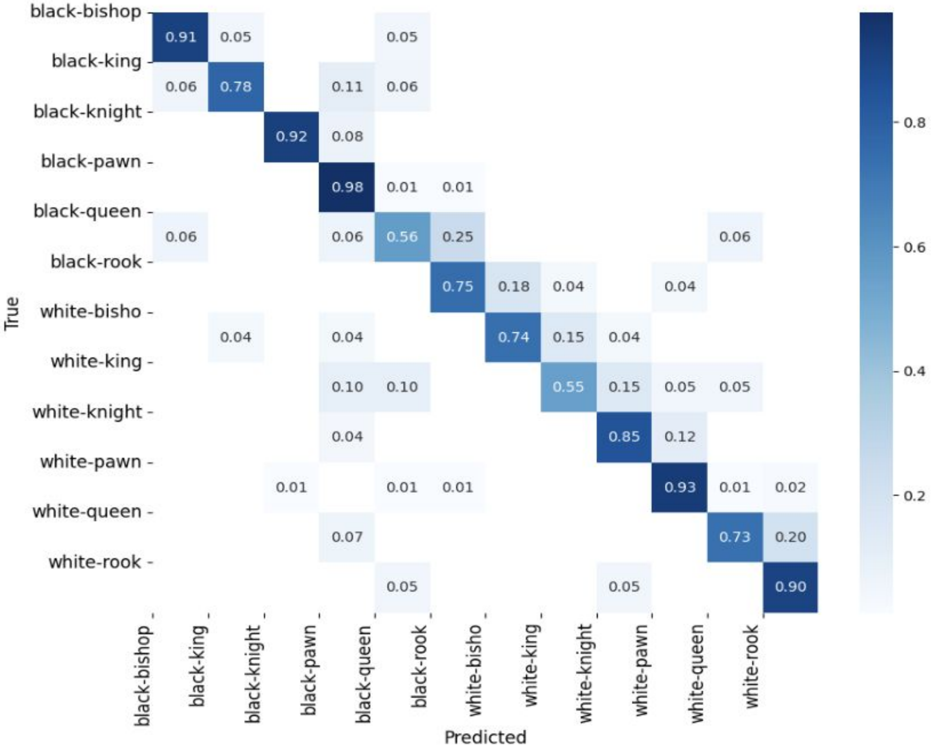


Lichess Board

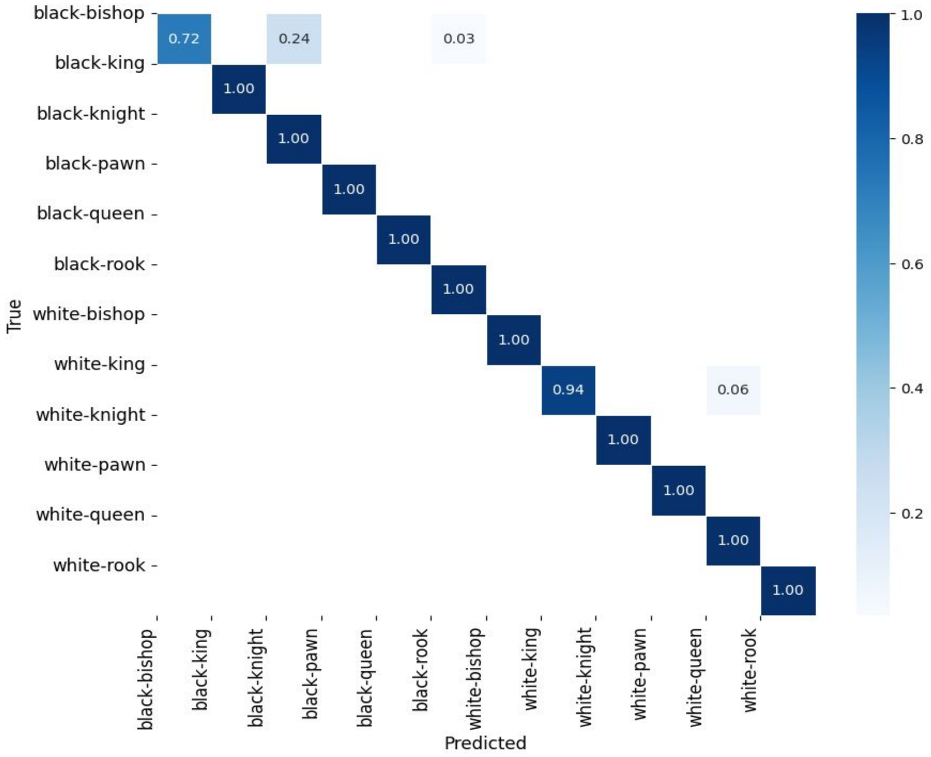
Experiments

- **YOLOv8** model used for **benchmarking** DETR's efficiency
- **Confusion matrix** computed for both models
- Calculated the Average Precision (**AP**) and Average Recall (**AR**) used as standard metrics
- Evaluation based on:
  - Performance of YOLOv8 compared to DETR model
  - Analysis of AP, AR and confusion matrix
- **Last transformer head** visualization

DETR Confusion Matrix



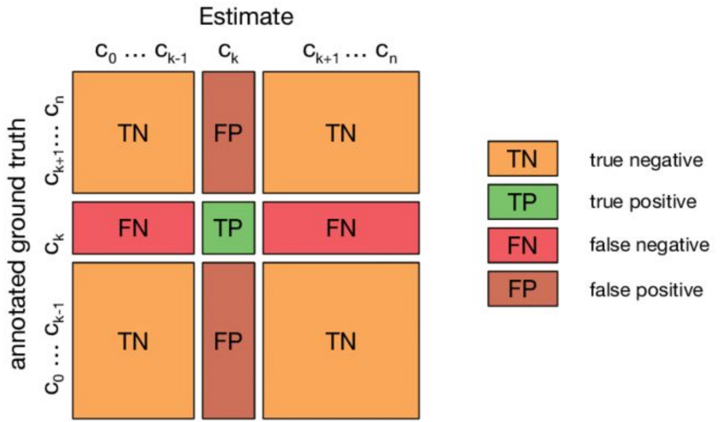
YOLO Confusion Matrix

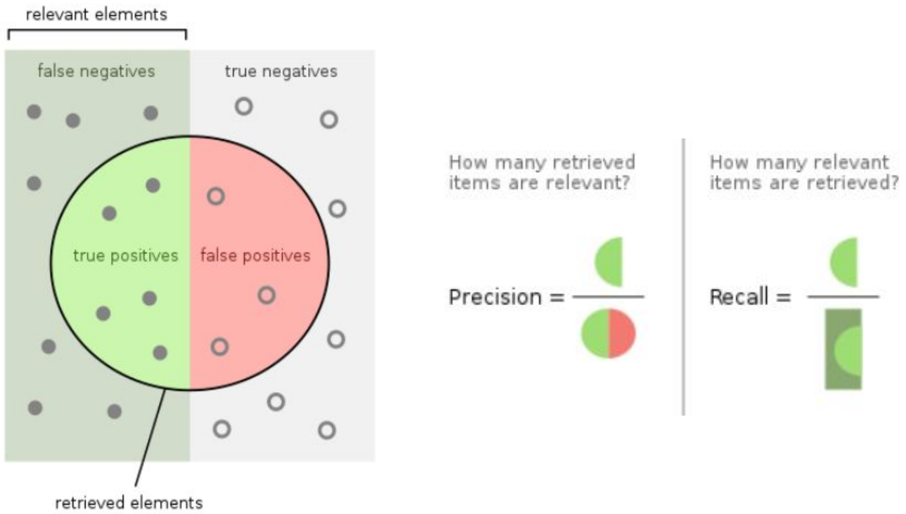


AP & AR

Average Precision: measures the probability of classifying a detection correctly averaged over all classes

Average Recall: measures the proportion of relevant items correctly detected by a model, averaged across all classes

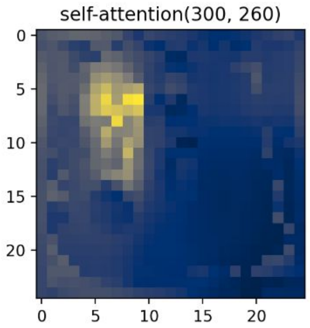


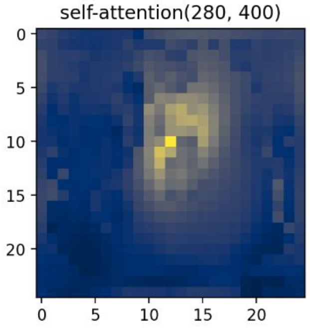


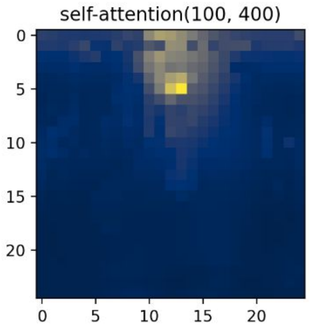
AP & AR

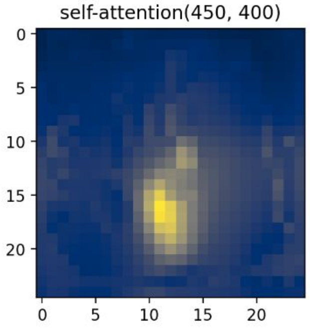
	DETR		YOLO	
	AP	**AR**	**AP**	**AR**
Pieces	0.75	0.75	**0.97**	**0.97**
Corners	**0.88**	1	0.81	1

Encoder self-attention weights

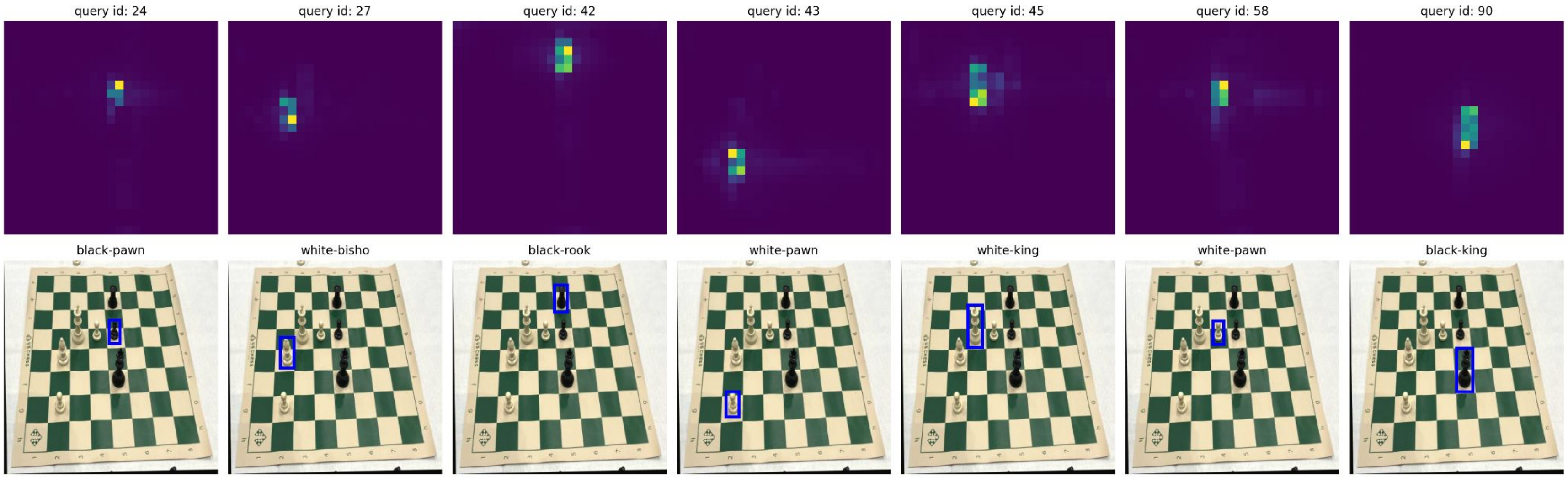








Attentions weights of the last decoder layer



Conclusions

- **Performance**:
  - DETR struggles to keep up with YOLO
  - Corner Detection is acceptable
- Equal Avg. Precision and Avg. Recall:
  - Could be because of very specific detection tasks (few classes)
- **Future Work**:
  - Hyperparameter tuning
  - Larger Datasets
  - Enhancements for robustness in failure to detect all corners
  - Enhancements for robustness in image transformation by rescaling the bounding boxes
  - Real-time detection using video feed
  - API integration with chess gaming software for real-life chess interaction