Project 3: Synthetic-to-Real Image Translation for Chessboard Rendering

Description:

In this project, you will train a chessboard image generation model whose goal is to transform synthetic chessboard renders into images that visually match real-world appearance while preserving the geometry and board layout (piece positions, camera pose, etc.).
This is an image-to-image translation task focused on closing the visual domain gap between synthetic and real images.

Unlike Project 2, which trains a classifier on synthetic images and evaluates its ability to generalize to real images, Project 3 is not about classification at all, the output here is an image (regression), not a set of labels.
In this project, your goal is to train a model that **translates synthetic chessboard renders into realistic-looking images**. Once trained, the model will enable you to generate entirely new board states synthetically and then transform them so they appear visually consistent with real data. (that's the test case in this task)

You will receive labeled frames with the corresponding chessboard state (piece-square positions only, occlusions are not labeled), as well as PGN labeled games.
The PGN files contain the full game state information, which can be used to easily generate additional frame level labels .(You have all board states from the PGN throughout the game, but you don't know which state corresponds to each frame.)

You will receive board-generation code (that I created) using Blender and PyBlender, along with a predefined set of 3D chess pieces and a chessboard. If you need additional synthetic variation (e.g., different piece designs), you may adapt the code and download more Blender chess sets (a website will be provided).

You need to install blender to your computer. https://www.blender.org/download/

Key Components:

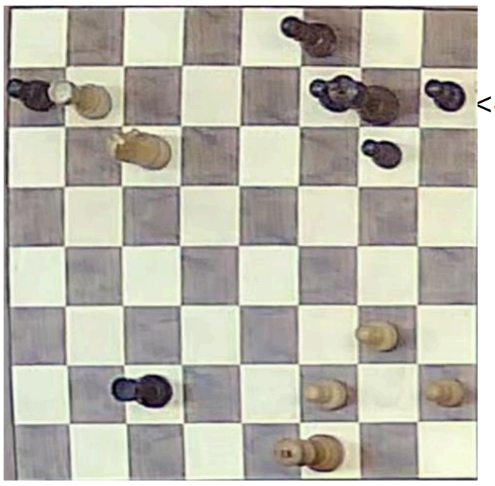
- Giving board generation code (that I created to some level) using Blender and PyBlender you will need to generate the data you need.
- Train a model that converts synthetic chessboard renders into realistic-looking images. Preserving geometric structure: piece identity, board layout, etc.
- Exploring image-to-image translation methods (e,g, GANs, CycleGAN, diffusion-based models, etc)
- Evaluating how well the translated images match the real domain visually (qualitatively and/or quantitatively)

- Some of the images have occluded squares (see examples in PDF of project 1). Is it possible to generate data without occlusion?
- Robustness to new game states.

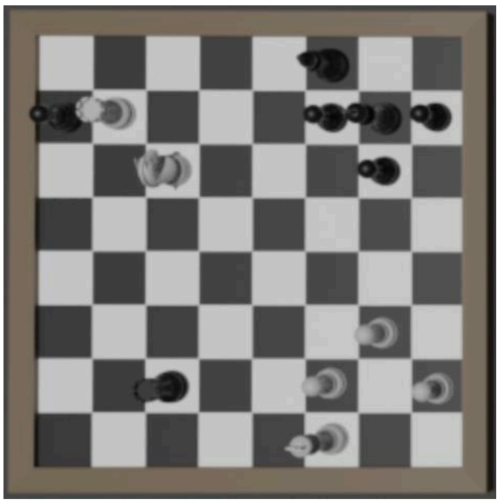
Additional notes:

1. **You don't have to** use the PGN games, but you might find it useful.
2. **You are allowed** to leverage the temporal nature of the video (i.e., consecutive frames) to generate additional labeled data from the PGN files.
For example, you can do so either by manually labeling frames or by applying any algorithm of your choice, classical methods, self-/unsupervised approaches, your own trained models, or off-the-shelf models that will label your data.
4. Your generation model <u>must not</u> rely on temporal information (e.g., what happened in previous or subsequent frames). Its only input is a single static image of the board, and its output is a realistic image.
5**.** Try to make your model generalize to new game states the best you can, be creative, train the model smartly.
6. **You are allowed** to download more 3D Chess assets, adjust my code to match it for rendering, and use it as more diverse synthetic data of chess boards and pieces if you find it useful. (e.g, I used this one https://www.blendswap.com/blend/29244)

Example:

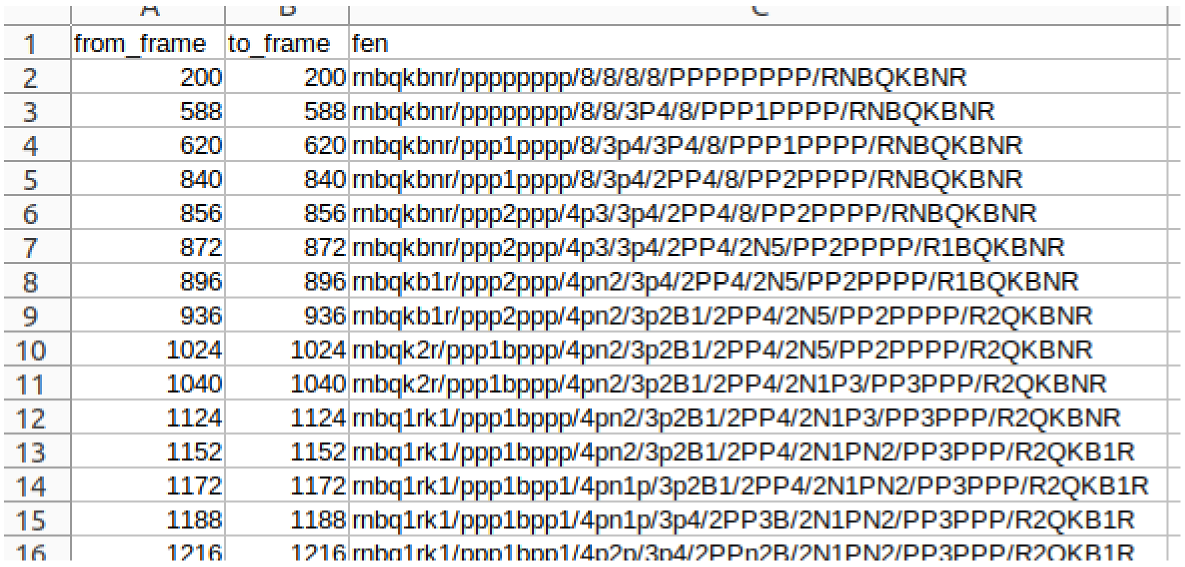


<-- Synthetic Image -->
using the ground
truth state of the
original image.



You will be provided with ZIP files for each game.
Each ZIP contains an **images/** folder and either a **game.csv** file or a **PGN** file with the corresponding labels.

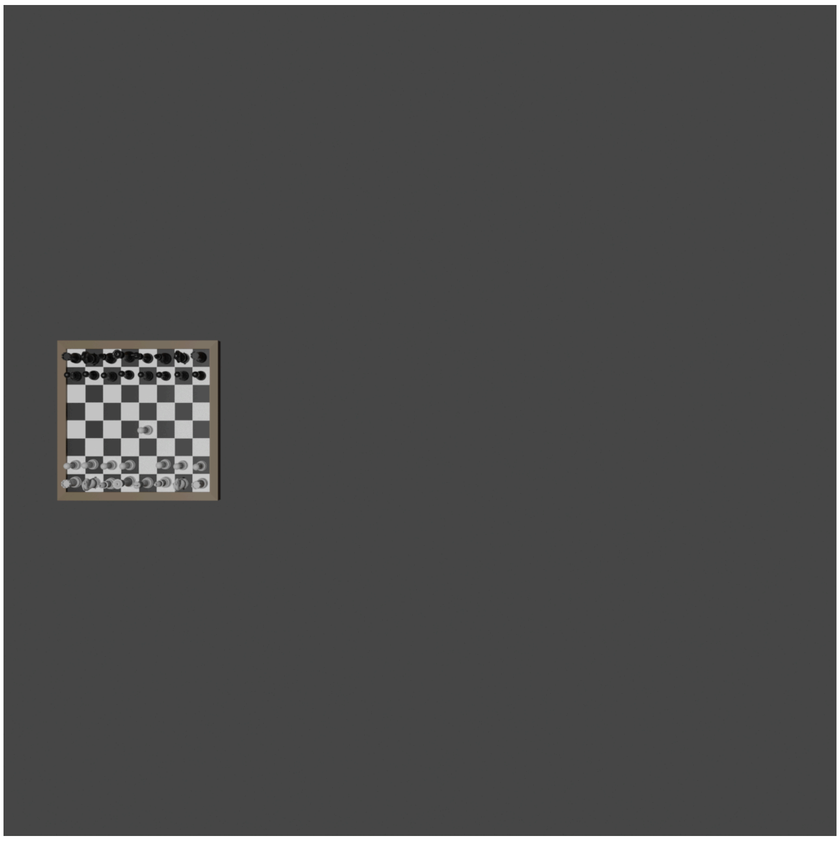
The **game.csv** labeling works as follows: for every distinct board state in the game, there is a specific frame that corresponds to that state.



	A	B	C
1	from_frame	to_frame	fen
2	200	200	rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR
3	588	588	rnbqkbnr/pppppppp/8/8/3P4/8/PPP1PPPP/RNBQKBNR
4	620	620	rnbqkbnr/ppp1pppp/8/3p4/3P4/8/PPP1PPPP/RNBQKBNR
5	840	840	rnbqkbnr/ppp1pppp/8/3p4/2PP4/8/PP2PPPP/RNBQKBNR
6	856	856	rnbqkbnr/ppp2ppp/4p3/3p4/2PP4/8/PP2PPPP/RNBQKBNR
7	872	872	rnbqkbnr/ppp2ppp/4p3/3p4/2PP4/2N5/PP2PPPP/R1BQKBNR
8	896	896	rnbqkb1r/ppp2ppp/4pn2/3p4/2PP4/2N5/PP2PPPP/R1BQKBNR
9	936	936	rnbqkb1r/ppp2ppp/4pn2/3p2B1/2PP4/2N5/PP2PPPP/R2QKBNR
10	1024	1024	rnbqk2r/ppp1bppp/4pn2/3p2B1/2PP4/2N5/PP2PPPP/R2QKBNR
11	1040	1040	rnbqk2r/ppp1bppp/4pn2/3p2B1/2PP4/2N1P3/PP3PPP/R2QKBNR
12	1124	1124	rnbq1rk1/ppp1bppp/4pn2/3p2B1/2PP4/2N1P3/PP3PPP/R2QKBNR
13	1152	1152	rnbq1rk1/ppp1bppp/4pn2/3p2B1/2PP4/2N1PN2/PP3PPP/R2QKB1R
14	1172	1172	rnbq1rk1/ppp1bpp1/4pn1p/3p2B1/2PP4/2N1PN2/PP3PPP/R2QKB1R
15	1188	1188	rnbq1rk1/ppp1bpp1/4pn1p/3p4/2PP3B/2N1PN2/PP3PPP/R2QKB1R
16	1216	1216	rnbq1rk1/ppp1bpp1/4p2p/3p4/2PPn2B/2N1PN2/PP3PPP/R2QKB1R

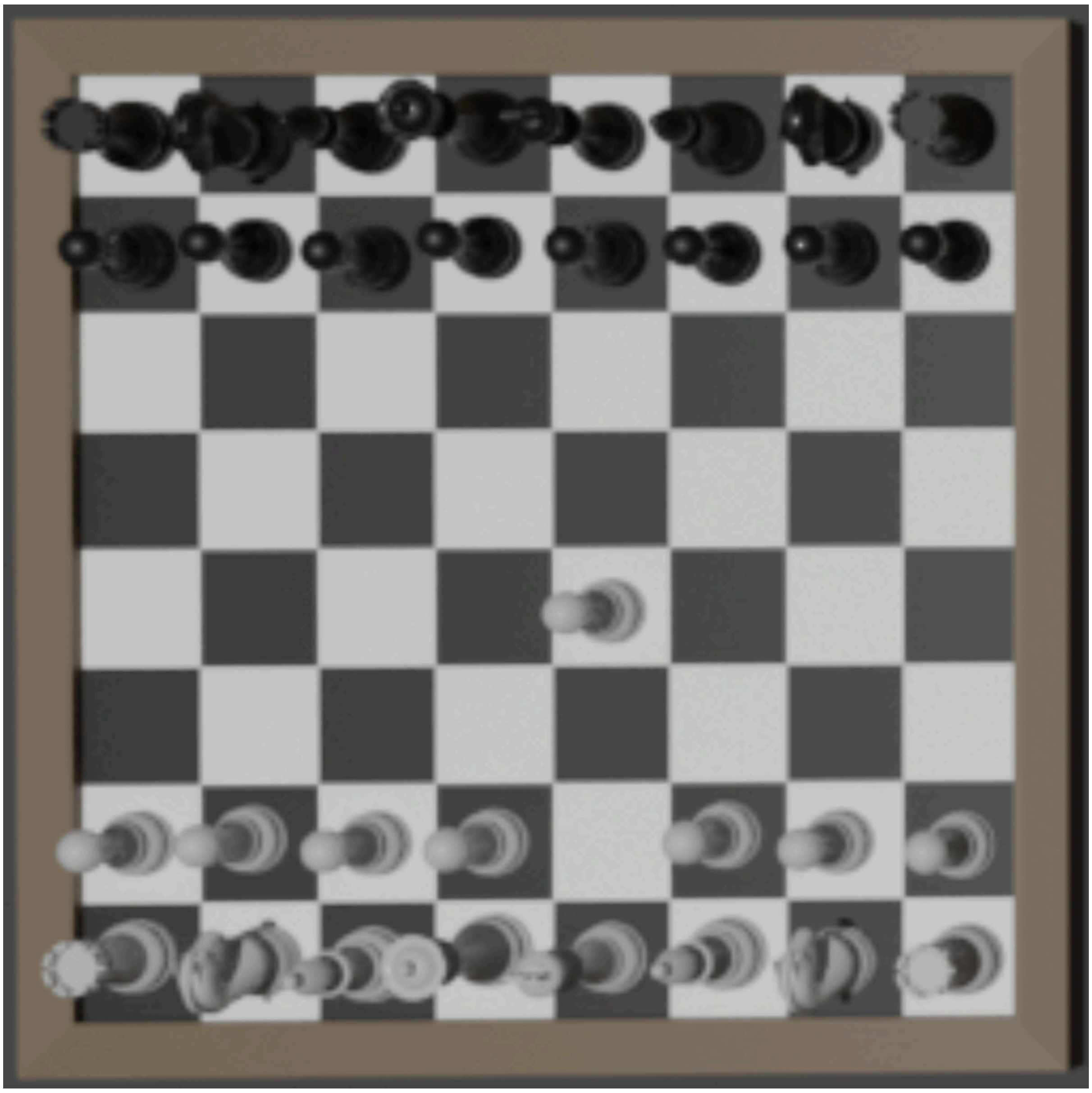
The **from_frame** and **to_frame** fields are identical. They both indicate the frame number in which the board appears in the state described by the **fen** column. (frames might have the same board state, so we chose only 1 for each state).

The code generation can produce the left board, the middle board and the right board. E.g, the left board.

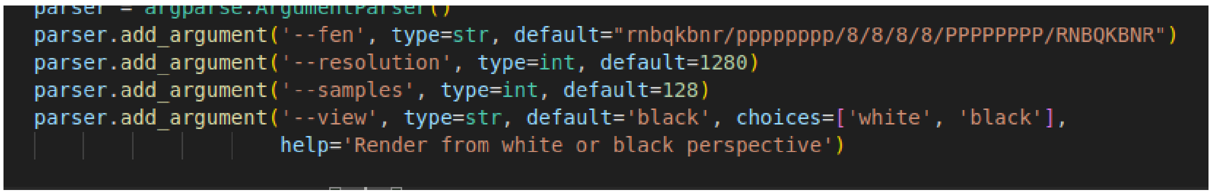


You should detect the board and then apply a perspective transformation so that the board fills the entire image. (same as done for the real images). Another way is to adjust my code in a way that will generate closeup boards, but be aware that we try to mimic the real data.

In example:



The code accepts a fen and generates 3 images (middle board, left board and right board). Adjust the code as needed, **the resolution that you choose will affect the time of rendering of each image. Start with a low resolution, like 800x800 or lower, it might be enough. (samples argument it's not the number of rendered images, it's also related to the quality), you might lower that value as well if you have to.**



```
parser = argparse.ArgumentParser()
parser.add_argument('--fen', type=str, default="rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR")
parser.add_argument('--resolution', type=int, default=1280)
parser.add_argument('--samples', type=int, default=128)
parser.add_argument('--view', type=str, default='black', choices=['white', 'black'],
                    help='Render from white or black perspective')
```

You also will be provided with the 3D chess asset, chess-set.blend.

How to run the code?

```
/home/roy/blender-5.0.0-linux-x64/blender chess-set.blend --background --python synthetic_chess.py -- --fen "8/2k5/2p5/2P5/8/3K4/8/6Q1"
```

“/home/roy/blender-5.0.0-linux-x64/blender” is the path to the downloaded blender, the data generation should be done in your computer and not via colab. (But the model training can and should be done via colab)

How to download python packages for blender’s python?

You can use:

```
/home/roy/blender-5.0.0-linux-x64/5.0/python/bin/python3.11 -m ensurepip
/home/roy/blender-5.0.0-linux-x64/5.0/python/bin/python3.11 -m pip install --upgrade pip
/home/roy/blender-5.0.0-linux-x64/5.0/python/bin/python3.11 -m pip install -r requirements.txt
```

(or any other package)

Good luck!