Chess Bot Using CNN with Lichess Dataset

1. Introduction

The project aims to create a chess bot capable of playing against human opponents by predicting optimal moves using Convolutional Neural Networks (CNNs). The bot is trained on the Lichess dataset, which provides real-world game data for model learning. The graphical user interface (GUI) is implemented using Pygame, while TensorFlow is used for model training and inference.

---

2. Objectives

- To develop a chess bot capable of making realistic and strategic moves.
- To train CNN models on the Lichess dataset for accurate move prediction.
- To integrate a GUI using Pygame for a user-friendly gaming experience.

---

3. Why Choose CNN for Chess Move Prediction?

In this project, CNNs were chosen due to their spatial compatibility and effectiveness in handling grid-like data, making them ideal for representing and analyzing chess boards.

3.1 Spatial Representation of the Chess Board

- The chess board is naturally a spatial grid (8×8).
- We represent the board as a 3D tensor with shape 8×8×128
    - 8 × 8: Board dimensions.

    - 12: Channels representing 12 piece types (pawn, knight, bishop, rook, queen, king × 2 colors).
- This is similar to image processing, where:
    - The board grid acts like image pixels.
    - The 12 channels resemble color channels (RGB).
- CNNs are ideal for such structured grid data, as they excel at detecting spatial patterns, essential for recognizing:

3.2 Local Feature Extraction

- CNNs use convolutional filters to detect local patterns like:
    - Piece formations.
    - Threatened squares.
    - Nearby board relationships.
- Benefits:
    - Prevents overfitting.
    - Improves training efficiency

4. Dataset

The Lichess dataset contains thousands of chess games, providing:

- Board states: Represented as pgn standard notation.
- Moves: Encoded in algebraic notation, serving as labels for the training data.
- The dataset was preprocessed into image-like representations for compatibility with CNNs.

---

5. Model Architecture

Two CNN models were developed to handle different aspects of move prediction:

- From-Model: Predicts the probability distribution of the piece to move.
- To-Model: Predicts the probability distribution of the destination square.
- The output probabilities of both models are multiplied to determine the best move.
- Models were trained using TensorFlow, with:
    - Categorical cross-entropy loss.
    - Adam optimizer for efficient gradient updates.

---

6. Integration and GUI

- Pygame Interface:
    - Displays the chessboard, pieces, and move history.
    - Allows interaction with the bot in a playable environment.
- Bot Decision Flow:

1. The board state is fed into the CNN models.
2. The From-Model predicts the piece to move.
3. The To-Model predicts the destination square.
4. The probabilities are multiplied, and the highest value determines the bot's move.
5. The move is rendered in the Pygame interface.

---

7. Results

- The bot successfully predicts and makes legal moves.
- The CNN models demonstrate reliable accuracy in selecting valid and effective moves.
- The GUI provides an interactive and visually appealing experience.

---

8. Challenges and Future Improvements

Challenges:

- Balancing accuracy and inference speed.
- Handling edge cases such as pawn promotion, castling, and en passant.
- Blind Spots occurs where model ignores a relatively simple best moves .

Future Improvments:

- Reinforcement learning: Adding RL techniques for more strategic decision-making.
- GUI enhancements:
    - Move highlights.
    - Move history tracking.
- Using LSTM:
    - Using memory we can retain bots prior information that can recognise intrinsive patterns.

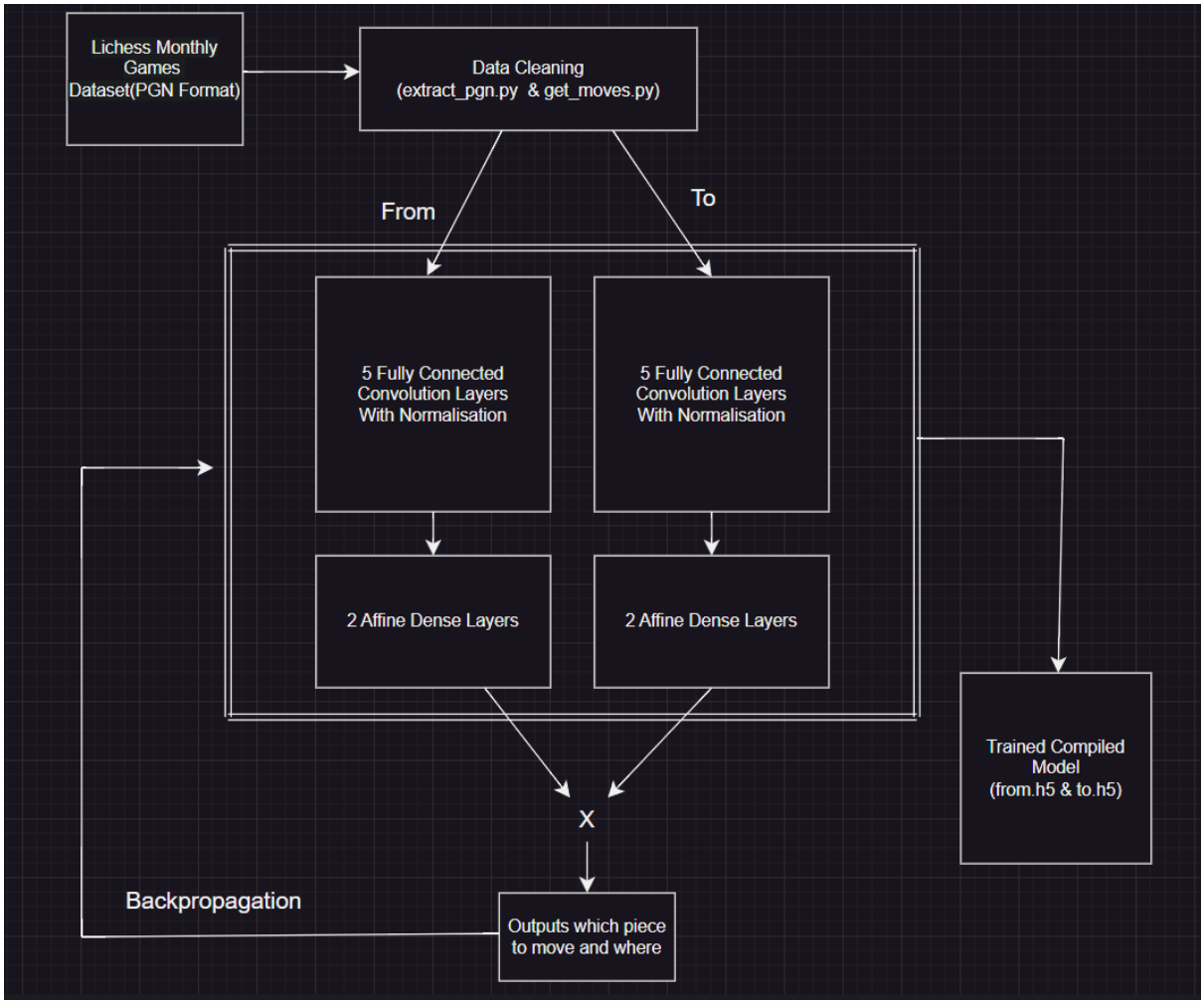
---

9. Conclusion

The chess bot demonstrates the effective use of CNNs for board game move prediction. The spatial compatibility of CNNs makes

them ideal for chess boards, while TensorFlow and Pygame allow for seamless model training and interaction. This project showcases the power of deep learning in board game AI, providing a solid foundation for further enhancements.

OverAll structure for the whole project can be denoted with this Diagram


Lichess Monthly Games Dataset(PGN Format)
Data Cleaning (extract_pgn.py & get_moves.py)
From
To
5 Fully Connected Convolution Layers With Normalisation
5 Fully Connected Convolution Layers With Normalisation
2 Affine Dense Layers
2 Affine Dense Layers
Trained Compiled Model (from.h5 & to.h5)
X
Backpropagation
Outputs which piece to move and where

Code for training .

Each conv function include filters, batch normalization and ReLu activation function

```
import tensorflow as tf

from model_parts import conv, affine

board = tf.keras.Input(shape=(8, 8, 12))
c1 = conv(x=board, filters=32)
c2 = conv(x=c1, filters=64)
c3 = conv(x=c2, filters=256)
c4 = conv(x=c3, filters=256, skip_connection=c3)
c5 = conv(x=c4, filters=256, skip_connection=c2)
c6 = conv(x=c5, filters=256, skip_connection=c1)
x = affine(x=c6, neurons=256)
x = affine(x=x, neurons=64)
x = affine(x=x, neurons=1)
x = tf.keras.layers.Softmax(axis=[1, 2])(x)

model = tf.keras.Model(inputs=[board,], outputs=x)

model.compile('adam', 'categorical_crossentropy')
```

The Main running component of the code looks as follows

```python
from tensorflow.keras.models import load_model

from_model = load_model(r'C:\Users\Shresth\vscode2\pyfiles\chess-ai\gui\models\1200-elo\from.h5', compile=True)
to_model = load_model(r'C:\Users\Shresth\vscode2\pyfiles\chess-ai\gui\models\1200-elo\to.h5', compile=True)

import pygame
import chess

from players import HumanPlayer, AIPlayer
from draw import draw_background, draw_pieces
import globals

pygame.init()

SCREEN_WIDTH = 700
SCREEN_HEIGHT = 600

win = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption('Chess')

board = chess.Board()

white = HumanPlayer(colour='white')
black = HumanPlayer(colour='black')
white_ai = AIPlayer(colour='white', from_model=from_model, to_model=to_model)
black_ai = AIPlayer(colour='black', from_model=from_model, to_model=to_model)
black = black_ai
fps_clock = pygame.time.Clock()

run = True
white_move = True
human_white = True
game_over_countdown = 50
```

```python
def reset():
    board.reset()
    global white_move
    white_move = True

    globals.from_square = None
    globals.to_square = None

while run:
    fps_clock.tick(30)

    draw_background(win=win)
    draw_pieces(win=win, fen=board.fen(), human_white=human_white)

    pygame.display.update()

    if board.is_game_over():
        if game_over_countdown > 0:
            game_over_countdown -= 1
        else:
            reset()
            game_over_countdown = 50
        continue

    if white_move and not human_white:
        white.move(board=board, human_white=human_white)
        white_move = not white_move

    if not white_move and human_white:
        black.move(board=board, human_white=human_white)
        white_move = not white_move

    events = pygame.event.get()

    for event in events:
        if event.type == pygame.QUIT:
            run = False
            pygame.quit()

        elif event.type == pygame.MOUSEBUTTONDOWN:
            x, y = pygame.mouse.get_pos()

            if 625 <= x <= 675 and 200 <= y <= 260: # Change sides
                human_white = not human_white
```

```
        else:
            reset()
            game_over_countdown = 50
        continue

    if white_move and not human_white:
        white.move(board=board, human_white=human_white)
        white_move = not white_move

    if not white_move and human_white:
        black.move(board=board, human_white=human_white)
        white_move = not white_move

    events = pygame.event.get()

    for event in events:
        if event.type == pygame.QUIT:
            run = False
            pygame.quit()

        elif event.type == pygame.MOUSEBUTTONDOWN:
            x, y = pygame.mouse.get_pos()

            if 625 <= x <= 675 and 200 <= y <= 260: # Change sides
                human_white = not human_white

                if human_white:
                    white = black
                    white.colour = 'white'
                    black = black_ai
                else:
                    black = white
                    black.colour = 'black'
                    white = white_ai

                reset()

            elif 630 <= x <= 670 and 320 <= y <= 360: # Reset
                reset()

        if white_move and human_white and white.move(board=board, event=event, human_white=human_white):
            white_move = not white_move

        elif not white_move and not human_white and black.move(board=board, event=event, human_white=human_white):
            white_move = not white_move
```

The Gui Looks like this :