MTL 782: Assignment 1

The King, the Queen, and the all-seeing Fish

Rohin Garg, Harshit Joshi, Ananya Sharma

February 27, 2025

Data Pre-processing I

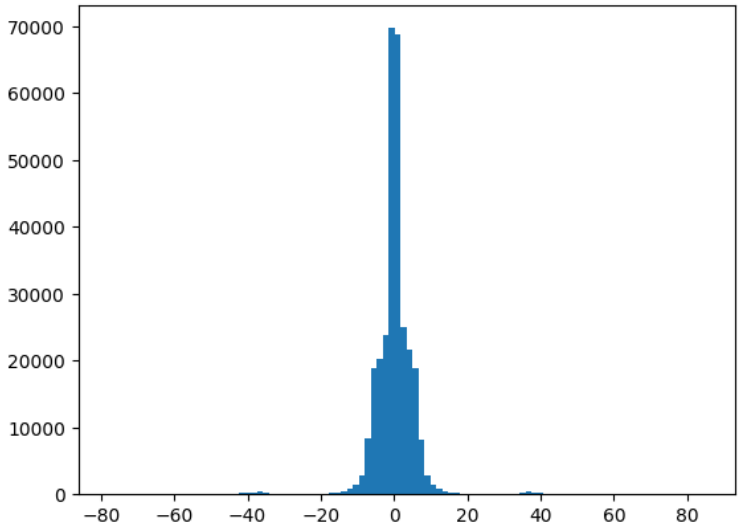
- After loading the data files into pandas dataframes, we selected the first 1,00,000 rows from each of the dataframes and concatenated them into a single dataframe 'df'.
- To parse the FEN strings, we split the data into separate columns.
- The next column is encoded numerically (-1 for black, 1 for white)
- A dictionary with binary indicators is created for castling rights
- Half and full moves also get encoded as integers.

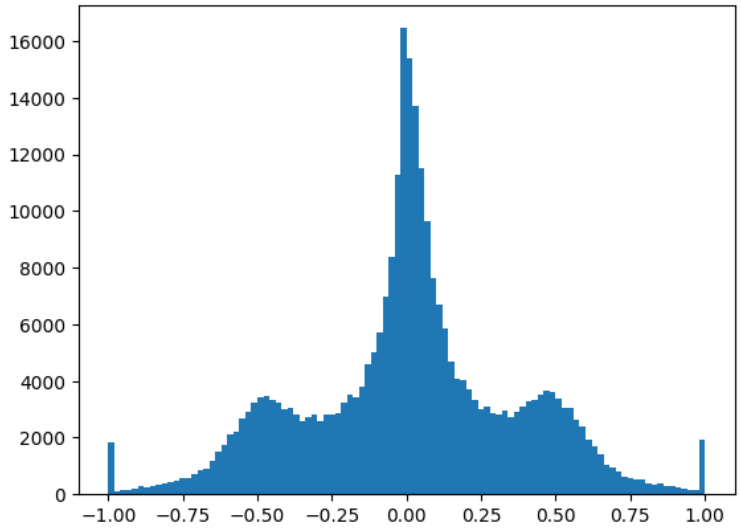
Data Pre-processing II

- We then created function 'numbw' that took a FEN string as an input and counted the number of each type of pieces.
- The function creates two dictionaries, 'blk' and 'wht', to store the number of each type piece for both white and black.
- Then it iterates through the FEN string and increments the counters as necessary.
- We applied this function to our data and added the output as columns to our dataframe.

Exploratory Data Analysis I

- To remove outliers from our data, we removed data whose Z-score was more than 3
- We plotted the distribution of the evaluated scores, first divided by 100 (since the evluations are in centipawns) and then normalised by dividing by 994.33977 and then applying the *tanh* function.x
- The strong clustering around 0 and symmetricity both persisted after normalisation, with small bumps corresponding to the ± 50 centipawn mark





Exploratory Data Analysis II

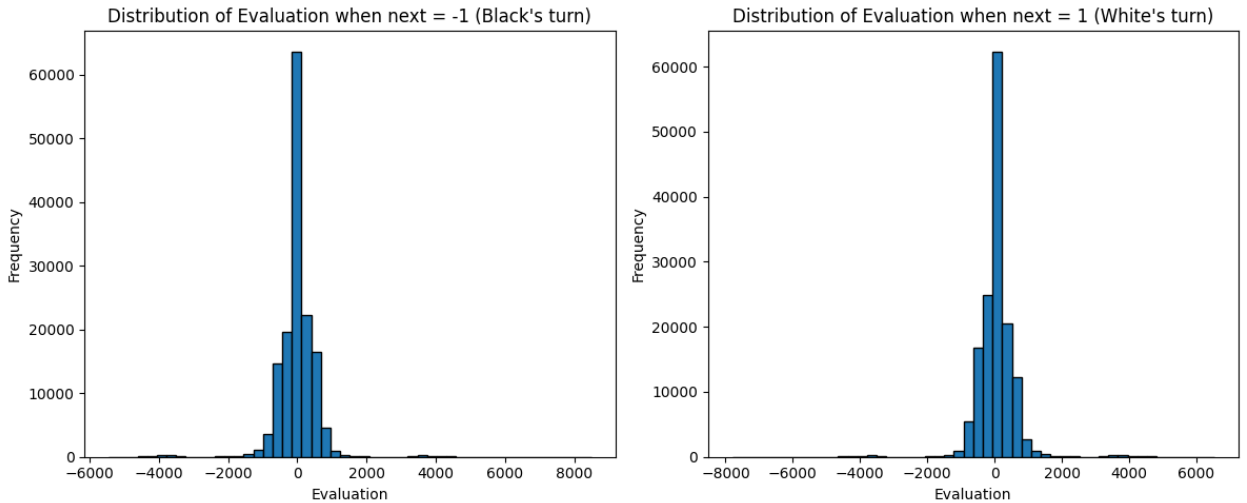
- Normalising the data using the *tanh* function helped us fit the data into the interval $[-1, 1]$, ensuring the data fits into a manageable interval
- Why was the scaling factor $\frac{1}{994.33977}$ chosen? Our rationale was that beyond a certain cutoff evaluation the marginal increase in the likelihood of victory goes to zero (for instance, not much difference between 15 and 500 pawn advantage)
- We wanted to find the optimal cutoff bound, for which we treated the scaling factor as a hyperparameter as it influences the distribution of the normalised data and is not known a priori

Exploratory Data Analysis III

- Thus, we could apply hyperparameter tuning and run for 20 cycles to find the optimal value of k
- Implemented it seperately using the Optuna software with Pytorch
- Searched for value in $[1, 1000]$ that minimised mean square error, got 994.33977
- Ran the model twice, got a similar value (differing by less than 1/10th)

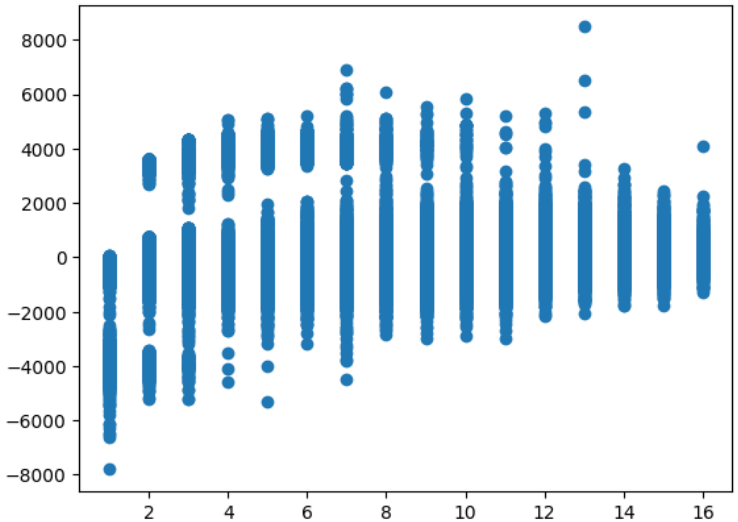
Exploratory Data Analysis IV

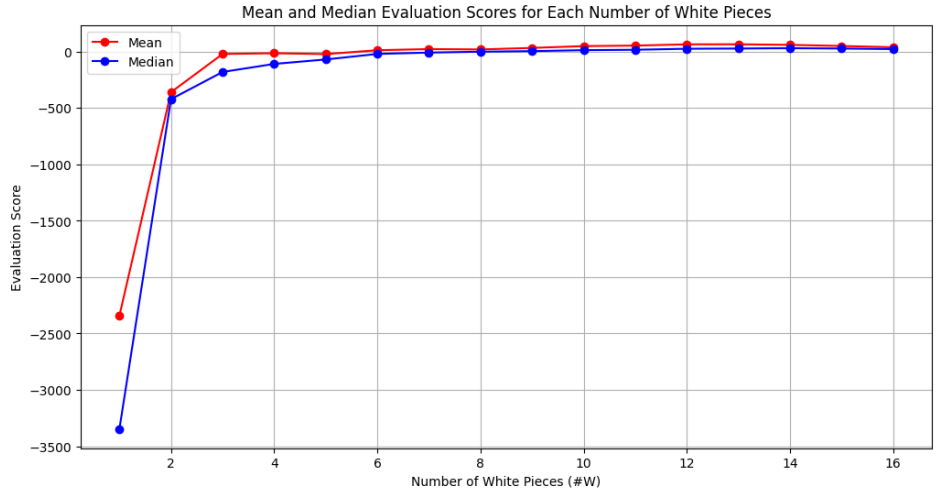
- Plotting the histograms for evaluations for when it was white's and black's turns to move respectively, both the distributions retained the sharp central peak at 0 as well as the symmetricity.
- For black's turn we obtain a slightly broader right tail, while for Whites turn we have a slightly more pronounced left tail.
- These indicate advantages that may be the result of the number of moves played



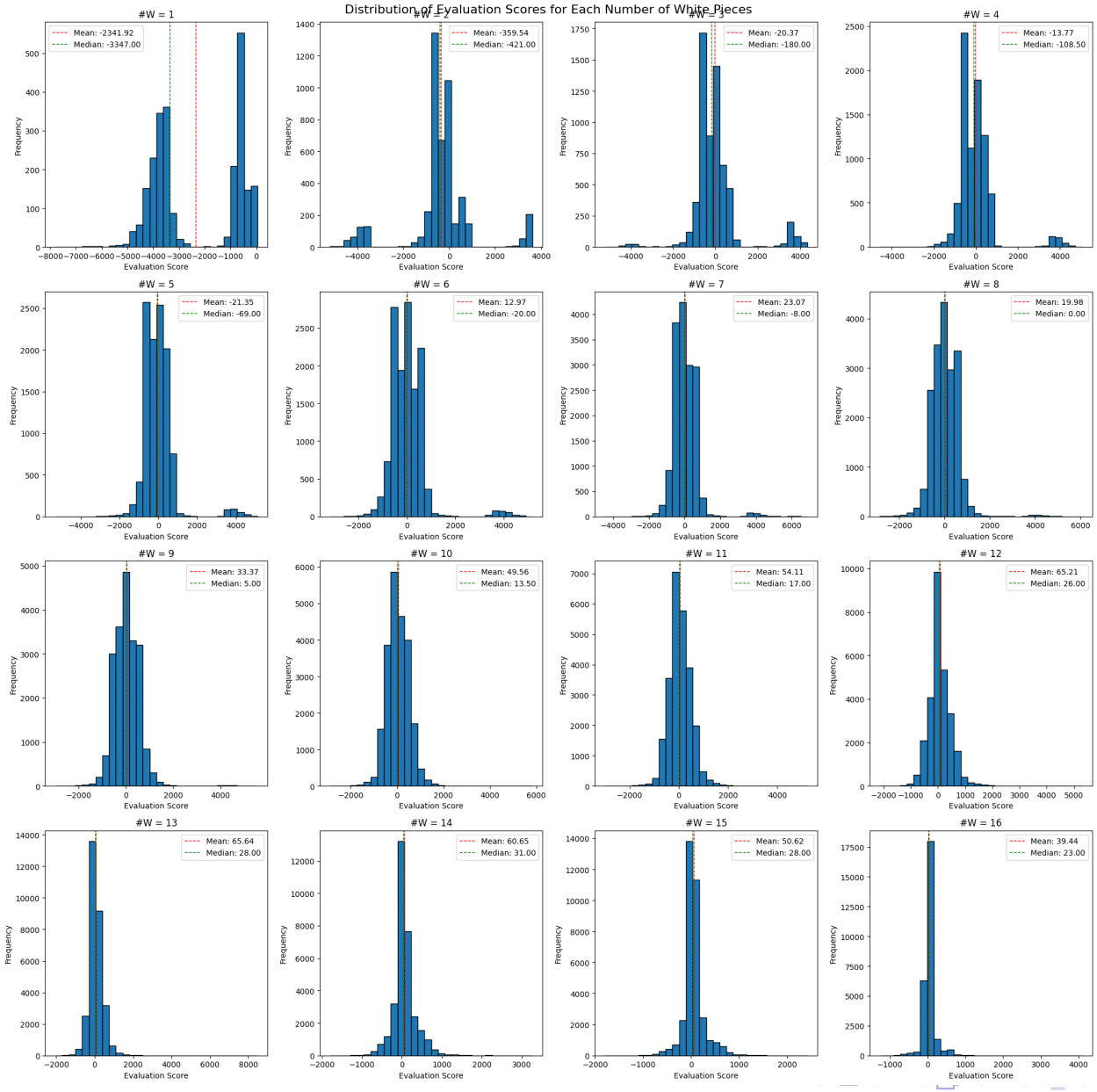
Exploratory Data Analysis V

- Whites evaluation scores decrease as the number of white pieces decreases (fewer pieces naturally imply a higher likelihood of losing).
- More extreme evaluations are observed with 10-14 pieces compared to positions with a full set.
- Makes sense since a full set likely $\implies$ openings in which there are lower chances of an advantage, whereas 10-14 pieces $\implies$ middlegame after execution of strategy (and hence reduced pieces)





Exploratory Data Analysis VI

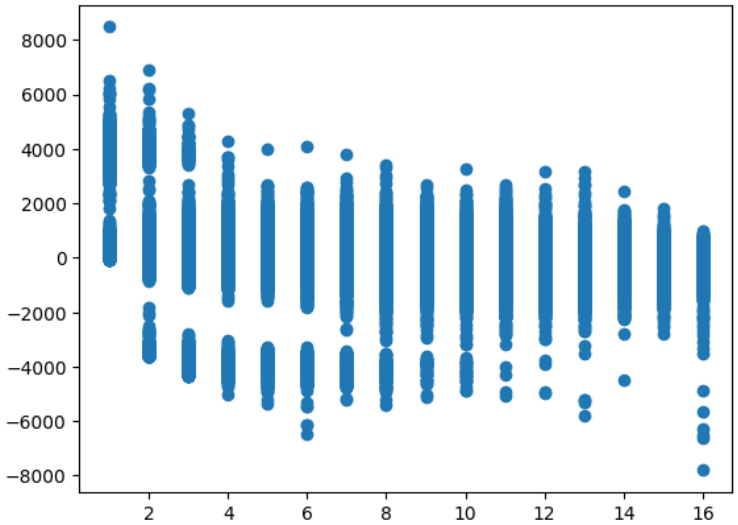


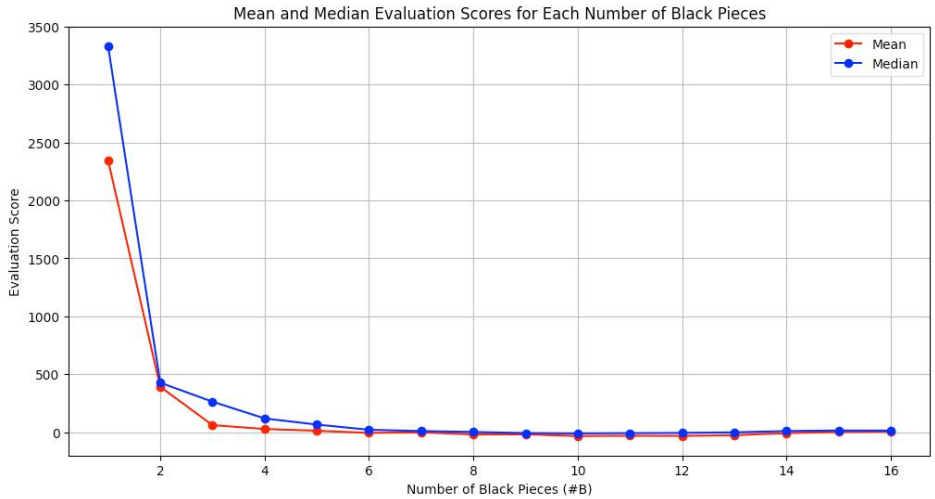
Exploratory Data Analysis VII

- Naturally, low pieces(1-4) for white give us a left-skewed distribution - most scores negative
- As the number of white pieces increases (5 to 12), the distributions become more symmetric - balanced
- At highest number of white pieces (13 to 16), the distributions remain symmetric but slightly narrower
- The highest mean evaluations occur in the 12-14 piece range (saw why earlier)
- At 8 white pieces, the most likely situation is a draw (effective evaluation 19.98, median evaluation of 0)

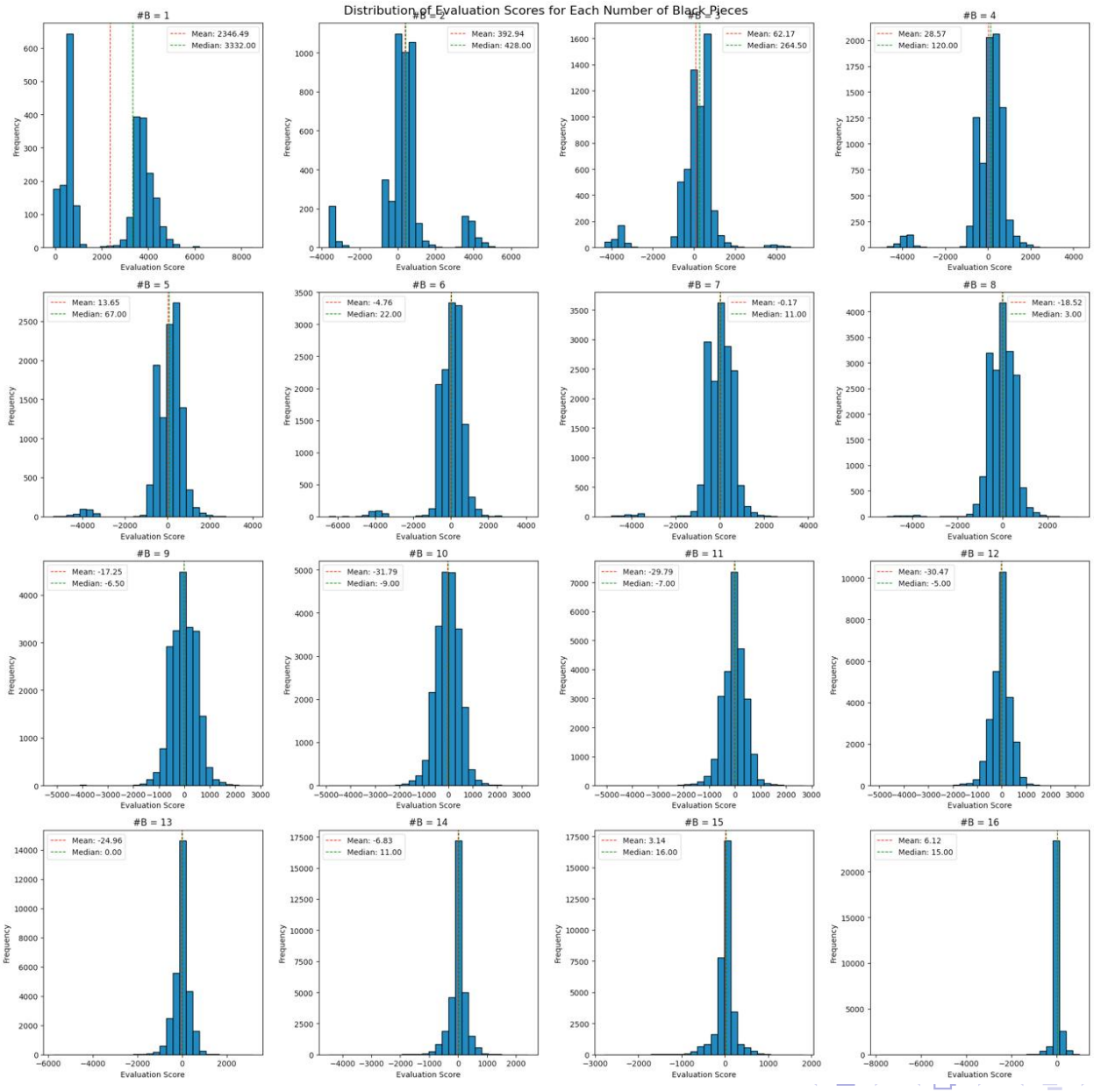
Exploratory Data Analysis VIII

- In a similar fashion black also see its evaluation scores decrease as the number of black pieces decreases
- Analogous to white black too sees a small bump in evaluation around the 12 piece as mark (for the same reasons as white)





Exploratory Data Analysis IX

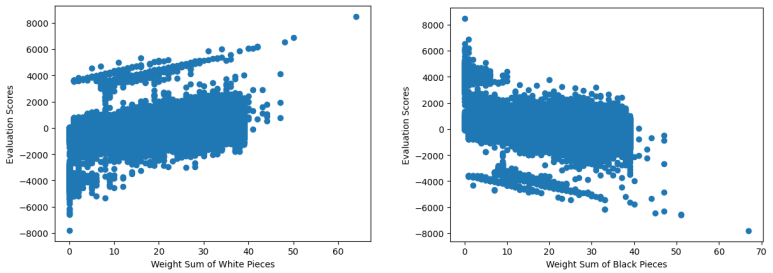


Exploratory Data Analysis X

- Individual histograms for black show more variability
- For more number of pieces the evaluations cluster around zero
- As the number of pieces reduces from 8 to 1, asymmetry develops: a large number of observations cluster around the mean followed by an island of evaluations at the extremes that slowly grow 1 Black piece is left the smaller island has a higher mode than the cluster around zero!

Exploratory Data Analysis XI

- Used linear regression to determine optimal piece weights for evaluation
- Trained separate model to find weights
- For white, we got Pawn: 129.80182169666838, Rook: 490.7844572280178, Knight: 316.9618465963061, Bishop: 354.8315553994522, Queen: 937.3200167799201, King: 0.0
- For black, we got Pawn: -133.23941840678827, Rook: -488.6501130856525, Knight: -309.8117720675912, Bishop: -347.6213057452584, Queen: -923.3755581785258, King: -3.410605131648481e-13
- As expected, evaluation scores for both white and black broadly increased as the weighted sum increased



Encoder 1

We used customed weights on the basis of our percieved importance and power of a chess piece.
We also use +ve and -ve connotation to white and black pieces respectively

- Model = { 'P': 1, 'N': 2, 'B': 5, 'R': 7, 'Q': 15, 'K': 10,
  'p': -1, 'n': -2, 'b': -5, 'r': -7, 'q': -15, 'k': -10 }
- "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR"

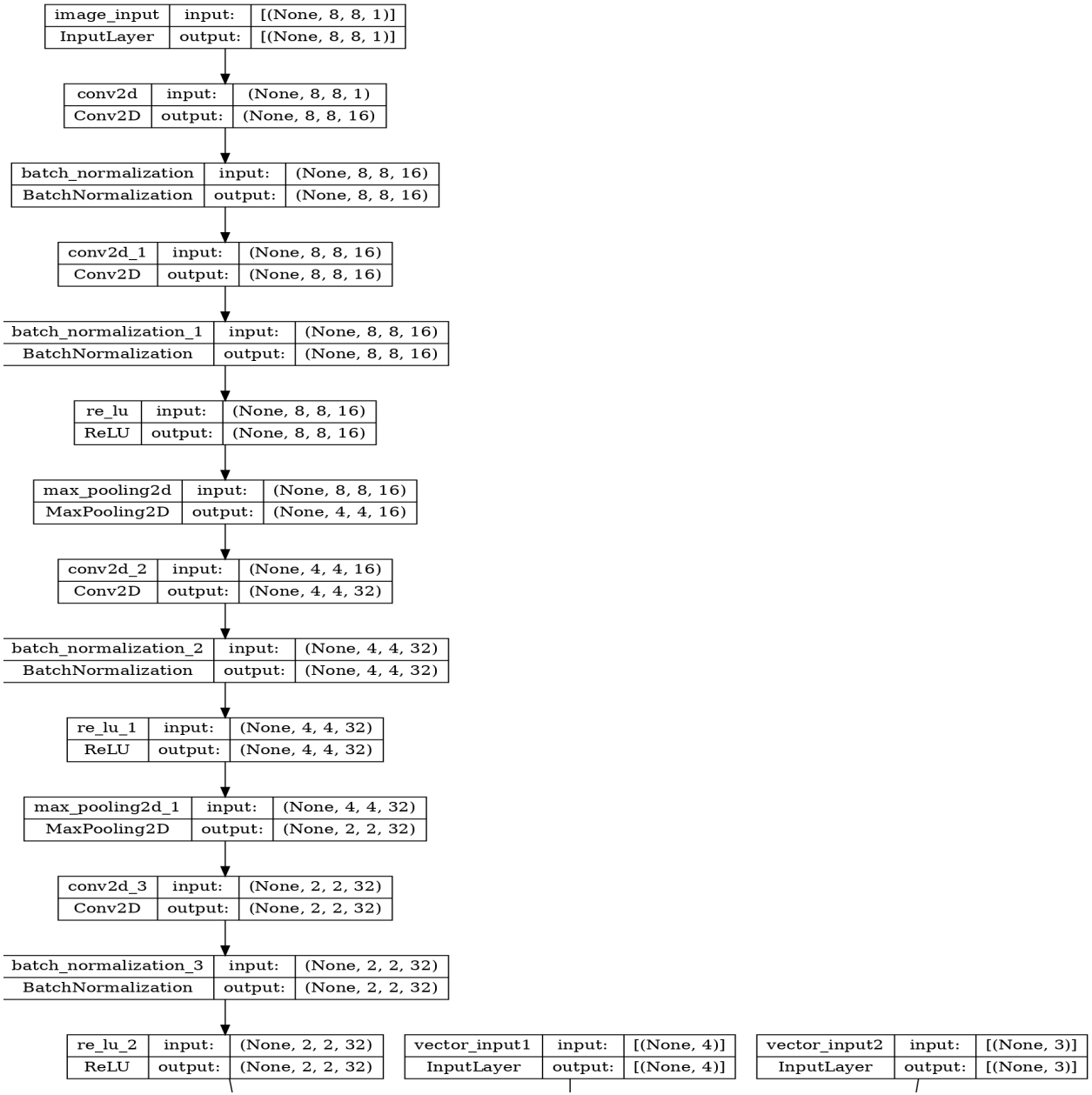
↓

[-7, -2, -5, -15, -10, -5, -2, -7,
-1, -1, -1, -1, -1, -1, -1, -1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1,
7, 2, 5, 15, 10, 5, 2, 7]

Base Model-CNN

- The Base Model we propose is a multi-input neural network that combines a 2D CNN for image input, along with 2 vector inputs to capture the remaining information in the given FEN Strings.
- The CNN input has dimensions (8,8,1).
- CNN consists of four 2D convolution layers. During the first two layers 16 3X3 filters are applied to the image followed by one layer of 32 3X3 filters and last layer of 32 2X2 filters.
- We increase the number of channels for this input to improve the model's representational capacity and hopefully better encode the positional nuances through various training data inputs

Base Model-CNN (Convolution Layers)



Base Model-CNN (Convolution Layers)

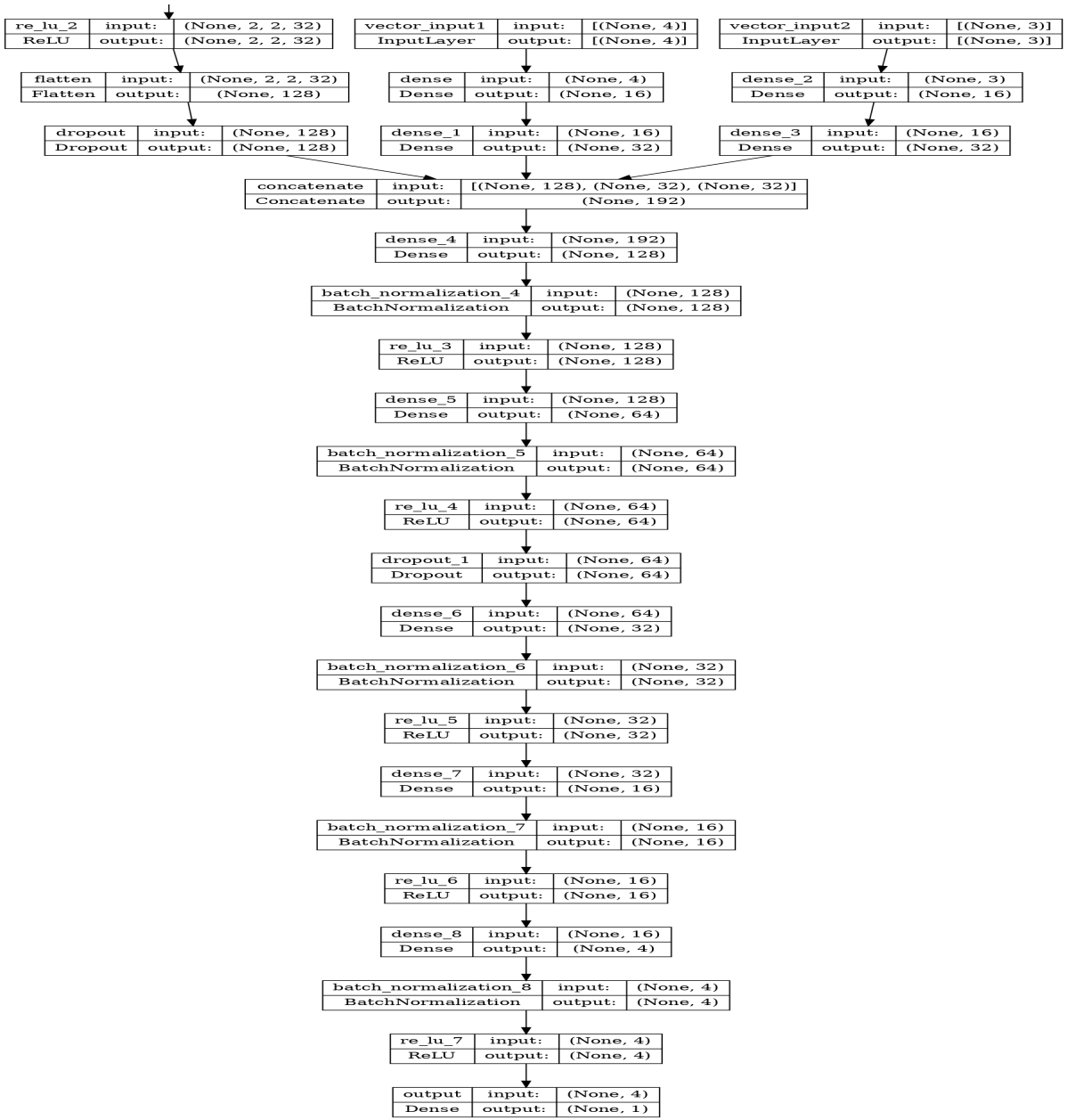
- ReLU activation function and MaxPooling is used on all the convolution layers.
- In addition to that Batch Normalization between all the convolution layers have been used to maintain consistent distributions across different layers.
- Final output of the convolution layers is (2,2,32) which we then flatten into a 128-dimensional vector.
- Dropout is applied to make the model more robust and to help reduce overfitting on the training data.

Base Model-CNN(Vector Inputs)

Two vector inputs are:

- The first vector represents the castling availability of K, k, Q, and q as a 4x1 vector of 0s and 1s (0 for not available, 1 for available).
- The second vector is (next, hm, fm), where:
  - "next" is a binary variable -1, +1 indicating the next active color.
  - "hm" is the number of Half-Moves.
  - "fm" is the number of Full-Moves.

Base Model-CNN (Vector Inputs)



Base Model-CNN(Vector Inputs)

- Each of these vector inputs is passed through a fully connected layer with 16 neurons.
- This encodes them into 16x1 vectors.
- We then pass these through another fully connected layer with 32 neurons.

Base Model-Concatenation

- At this point, we have 128x1, 32x1, and 32x1 vectors representing the image input and the two vector inputs, respectively.
- These are concatenated to form a 192x1 vector.
- The concatenated vector is then passed through several dense layers to reduce the dimensionality to 4x1, with batch normalization applied and ReLU as the activation function.
- This 4x1 vector is then connected to a single output neuron (via a linear combination), which provides the final output.

Encoder 2

Assigned unique weight from 0 to 11 to each piece.

- Model = { 'P':0, 'N':1, 'B':2, 'R':3, 'Q':4, 'K':5,
  'p':6, 'n':7, 'b':8, 'r':9, 'q':10, 'k':11 }
- "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR"

$$\downarrow$$

[9, 7, 8, 10, 11, 8, 7, 9,
6, 6, 6, 6, 6, 6, 6, 6,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
3, 1, 2, 4, 5, 2, 1, 3]

Encoder 3

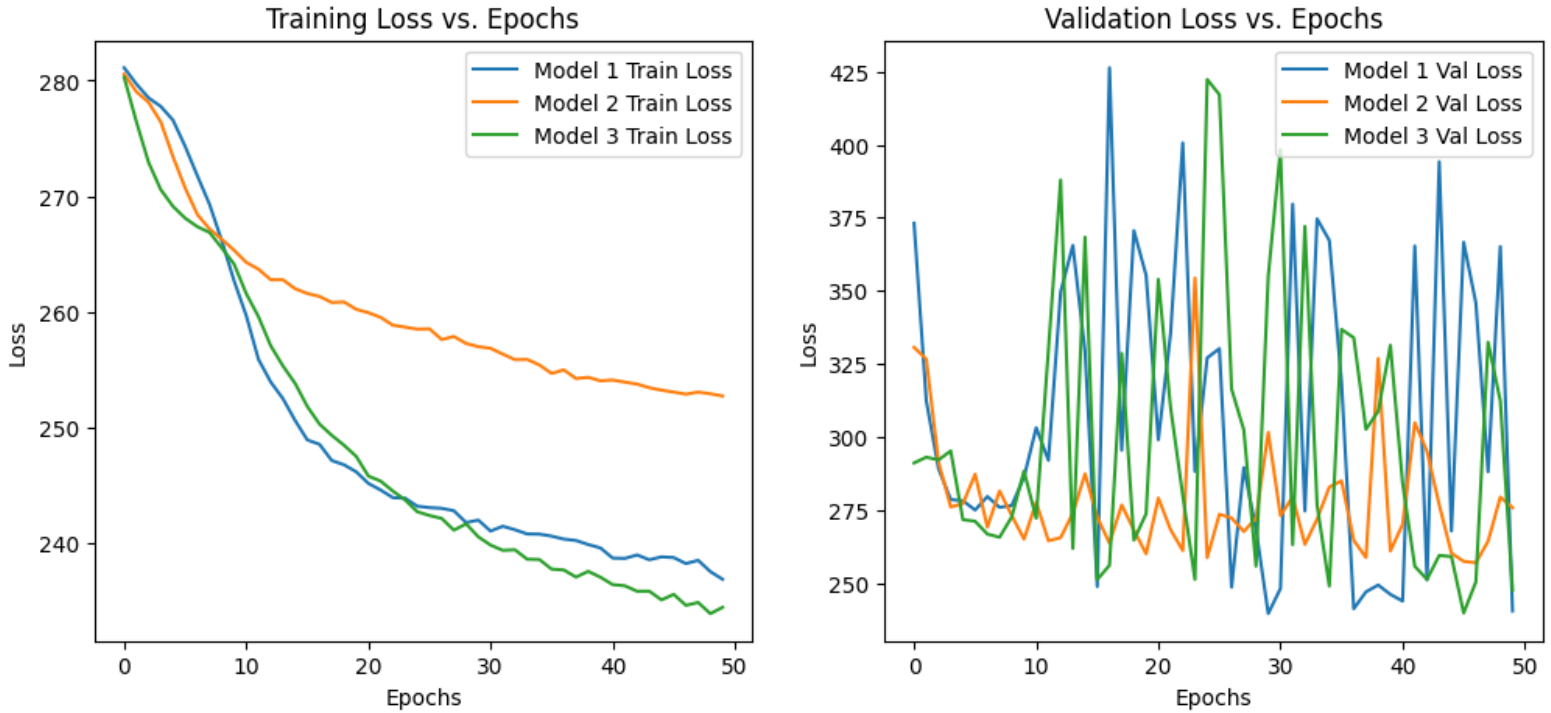
Modified encoder 1 by significantly increasing the weight of the king

- Model = { 'P': 1, 'N': 2, 'B': 5, 'R': 7, 'Q': 15, 'K': 100,
  'p': -1, 'n': -2, 'b': -5, 'r': -7, 'q': -15, 'k': -100 }
- "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR"

↓

[-7, -2, -5, -15, -100, -5, -2, -7,
-1, -1, -1, -1, -1, -1, -1, -1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1,
7, 2, 5, 15, 100, 5, 2, 7]

2d Model Comparison



- Encoder 3 performed better for training loss.
- For the validation loss, all three encodings ocsillate wildly, we consider Encoder 3 as it has the minimum best validation loss as compared to the rest.

Encoder 4

To further improve our FEN-to-image conversion approach, we explored encoding FEN strings into 3D arrays. We assigned mobility-based vectors for each piece, with positive values for white pieces and negative values for black pieces.

- Model = { 'r': -[0, 8, 0, 8, 0, 8, 0, 8],
  'n': -[2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5],
  'b': -[8, 0, 8, 0, 8, 0, 8, 0],
  'q': -[8, 8, 8, 8, 8, 8, 8, 8],
  'k': -[1, 1, 1, 1, 1, 1, 1, 1],
  'p': -[0, 0, 0, 0, 0, 1, 0, 0],
  'R': [0, 8, 0, 8, 0, 8, 0, 8],
  'N': [2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5],
  'B': [8, 0, 8, 0, 8, 0, 8, 0],
  'Q': [8, 8, 8, 8, 8, 8, 8, 8],
  'K': [1, 1, 1, 1, 1, 1, 1, 1],
  'P': [0, 0, 0, 0, 0, 1, 0, 0] }

Encoder 4 (contd)

- The CNN input image has dimensions (8,8,8).
- "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR"

$$\downarrow$$

$$R_1 = \begin{matrix} -0. & -8. & -0. & -8. & -0. & -8. & -0. & -8. \\ -2.5 & -2.5 & -2.5 & -2.5 & -2.5 & -2.5 & -2.5 & -2.5 \\ -8. & -0. & -8. & -0. & -8. & -0. & -8. & -0. \\ -8. & -8. & -8. & -8. & -8. & -8. & -8. & -8. \\ -1. & -1. & -1. & -1. & -1. & -1. & -1. & -1. \\ -8. & -0. & -8. & -0. & -8. & -0. & -8. & -0. \\ -2.5 & -2.5 & -2.5 & -2.5 & -2.5 & -2.5 & -2.5 & -2.5 \\ -0. & -8. & -0. & -8. & -0. & -8. & -0. & -8. \end{matrix}$$

$$R_2 = \begin{matrix} -0. & -0. & -0. & -0. & -0. & -1. & -0. & -0. \\ -0. & -0. & -0. & -0. & -0. & -1. & -0. & -0. \\ -0. & -0. & -0. & -0. & -0. & -1. & -0. & -0. \\ -0. & -0. & -0. & -0. & -0. & -1. & -0. & -0. \\ -0. & -0. & -0. & -0. & -0. & -1. & -0. & -0. \\ -0. & -0. & -0. & -0. & -0. & -1. & -0. & -0. \\ -0. & -0. & -0. & -0. & -0. & -1. & -0. & -0. \\ -0. & -0. & -0. & -0. & -0. & -1. & -0. & -0. \end{matrix}$$

Encoder 4 (contd)

$$R_3 = R_4 = R_5 = R_6 = \begin{matrix} 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \end{matrix}$$

$$R_7 = \begin{matrix} 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. \end{matrix}$$

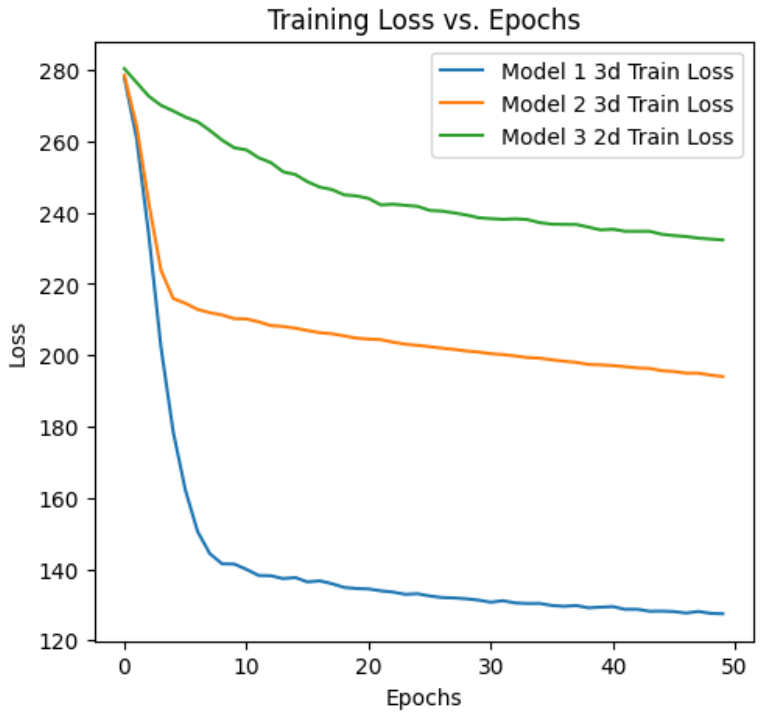
$$R_8 = \begin{matrix} 0. & 8. & 0. & 8. & 0. & 8. & 0. & 8. \\ 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 \\ 8. & 0. & 8. & 0. & 8. & 0. & 8. & 0. \\ 8. & 8. & 8. & 8. & 8. & 8. & 8. & 8. \\ 1. & 1. & 1. & 1. & 1. & 1. & 1. & 1. \\ 8. & 0. & 8. & 0. & 8. & 0. & 8. & 0. \\ 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 \\ 0. & 8. & 0. & 8. & 0. & 8. & 0. & 8. \end{matrix}$$

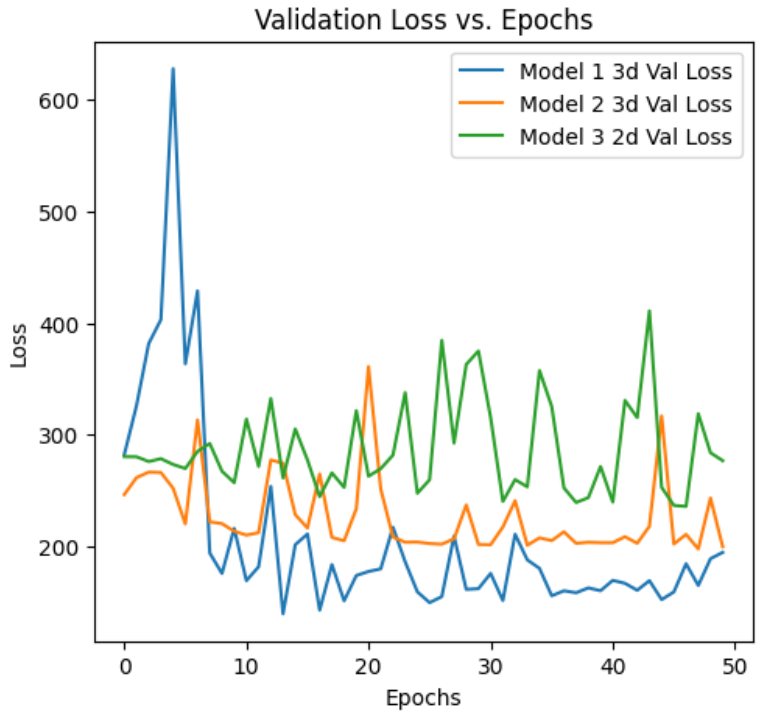
Encoder 5

We scaled the mobility vectors by conventional chess piece values, assigning higher weights to more valuable pieces

- Model = { 'r': -5*[0, 8, 0, 8, 0, 8, 0, 8],
  'n': -3*[2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5],
  'b': -3*[8, 0, 8, 0, 8, 0, 8, 0],
  'q': -9*[8, 8, 8, 8, 8, 8, 8, 8],
  'k': -10*[1, 1, 1, 1, 1, 1, 1, 1],
  'p': -[0, 0, 0, 0, 0, 1, 0, 0],
  'R': 5*[0, 8, 0, 8, 0, 8, 0, 8],
  'N': 3*[2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5],
  'B': 3*[8, 0, 8, 0, 8, 0, 8, 0],
  'Q': 9*[8, 8, 8, 8, 8, 8, 8, 8],
  'K': 10*[1, 1, 1, 1, 1, 1, 1, 1],
  'P': [0, 0, 0, 0, 0, 1, 0, 0] }

3d Model Comparison





- Encoder 4 performed better for training loss.
- For validation loss, after high loss initially, Encoder 4 settled down at the lowest level.
- Encoder 4 performed the best comparatively.

Testing

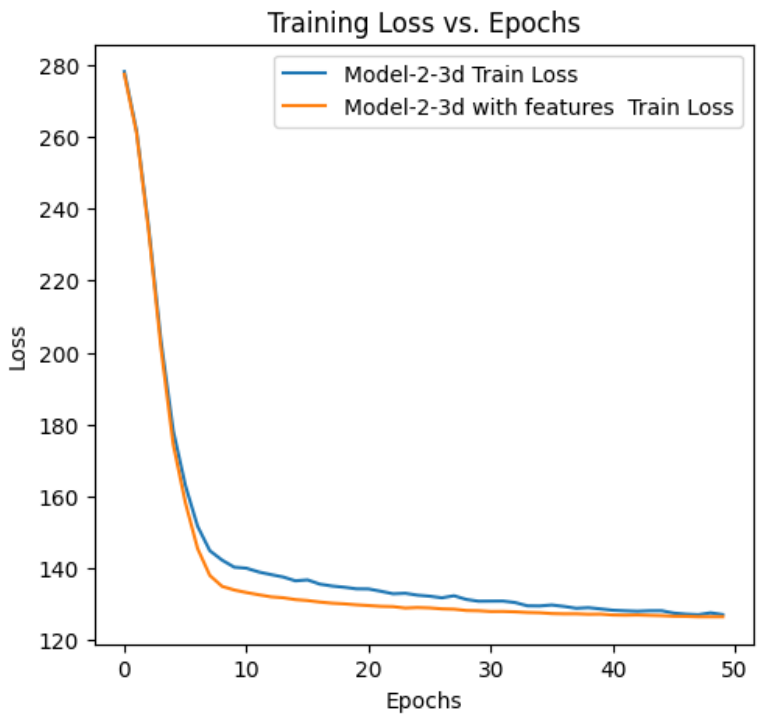
- Till now we have finalized our Base Model. To summarize, this model takes in a (8,8,8) representation of our chess board along with 2 additional vectors as inputs. The information we use is everything we can extract from the given FEN notations, and we utilize it fully.
- Next, we want to train various model to understand whether out handcrafted features are of any use or not, we also check for less data more epochs and more data less epochs and finally for different output (Y) encoding with different precision.

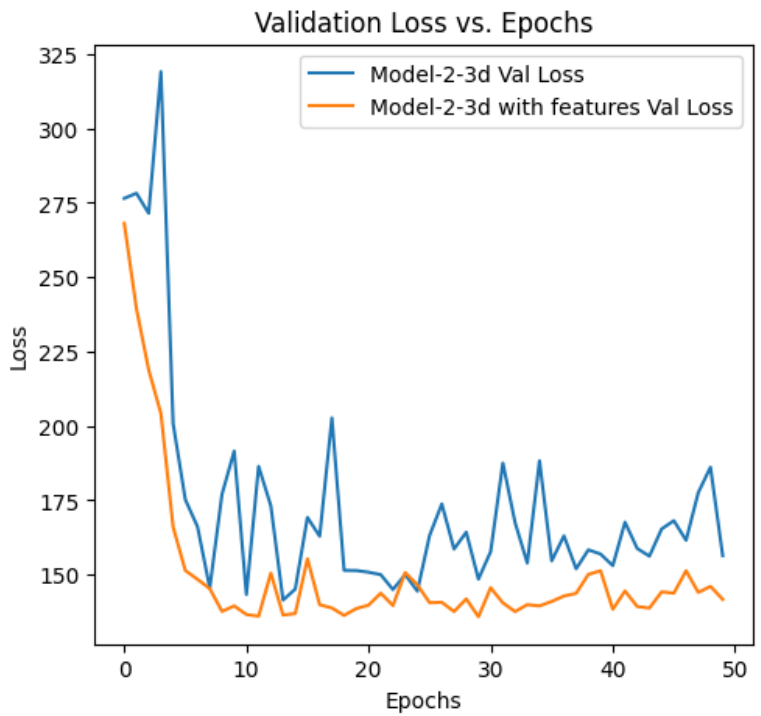
Testing

- for this we perform various trainings. To keep the comparison fair, we will fix the number of epochs, optimizer, and hyperparameters, unless stated otherwise.
  - **Data Fitting:**
    - Number of epochs: 50
    - Batch size: 2048
  - **Train-Test Split:**
    - Test size: 0.2
    - Random state: 42
  - **Optimizer:** AdamW from `tf.keras.optimizers` with:
    - Learning rate: 1e-2
    - Weight decay: 1e-3
  - **Model Metrics:**
    - Mean Square Error
    - Mean Absolute Error

Test-1 (HandCrafted Features)

- we train a new model to also account for our extra two features (wtdb and wtdw), this vector is also converted to a 32x1 vector and concatenated, the rest of the model remains the same, original y values .





- Clearly our features improve the model accuracy

Test-2 (Output Precision)

- Training the new model with features on two different output encodings:
  - First encoding uses the same y as before.
  - Second encoding uses:

$$y \leftarrow \tanh\left(\frac{\mathrm{y}}{994.33977}\right) \text{ rounded to 10 decimal places}$$

- Since the output is precise up to the 10th decimal place, the model is readjusted to use float32 throughout layers.

Custom Loss Function

- We design a custom loss function to ensure the outputs are comparable:

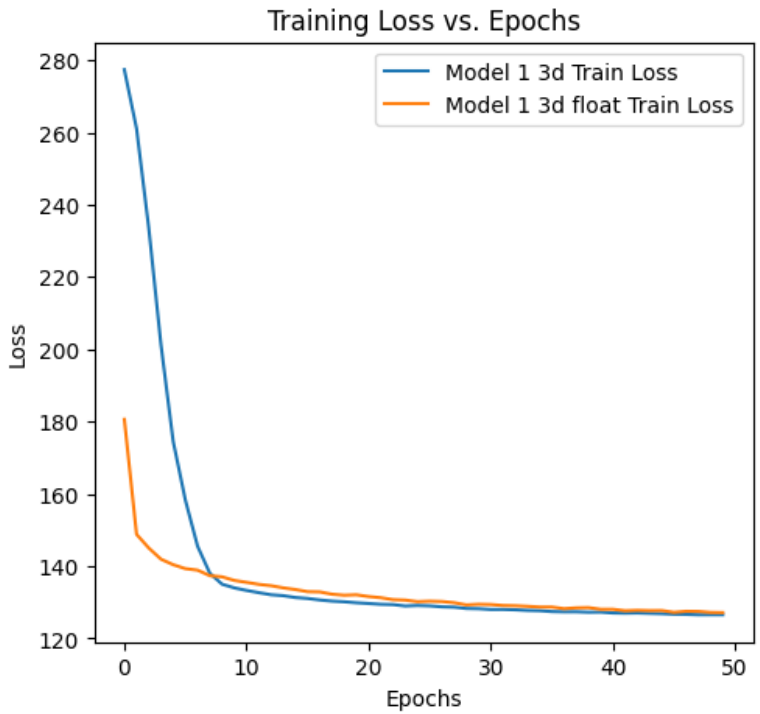
$$\text{customloss}(a, b)$$

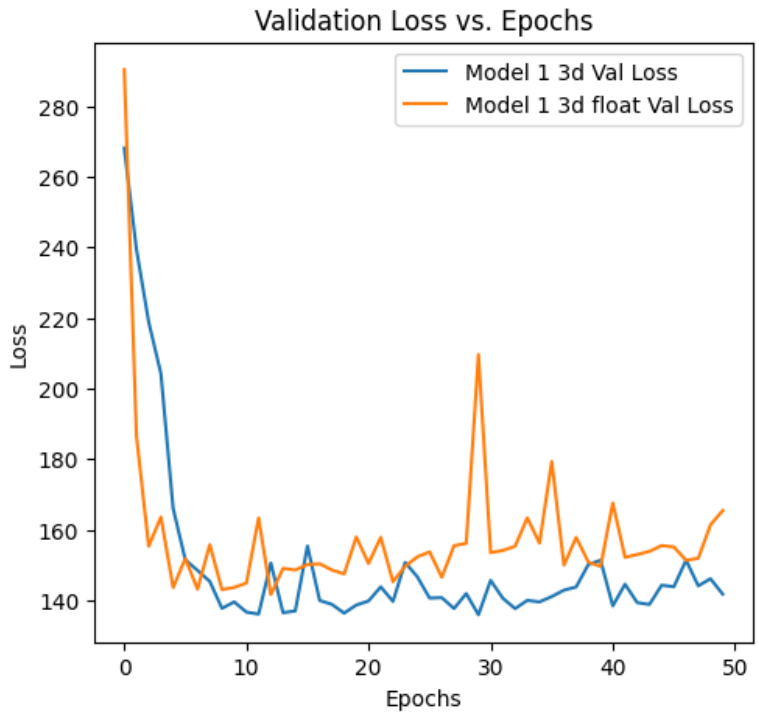
$$b \leftarrow 994.33977 \cdot \text{arctanh}(b)$$

$$a \leftarrow 994.33977 \cdot \text{arctanh}(a)$$

$$\text{return mean}\left(|b - a|\right)$$

Model Performance

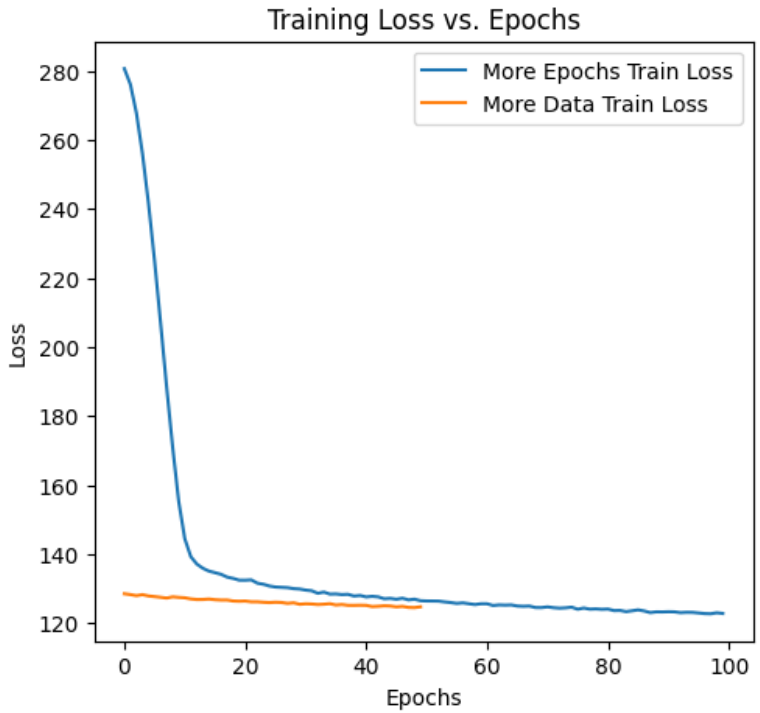


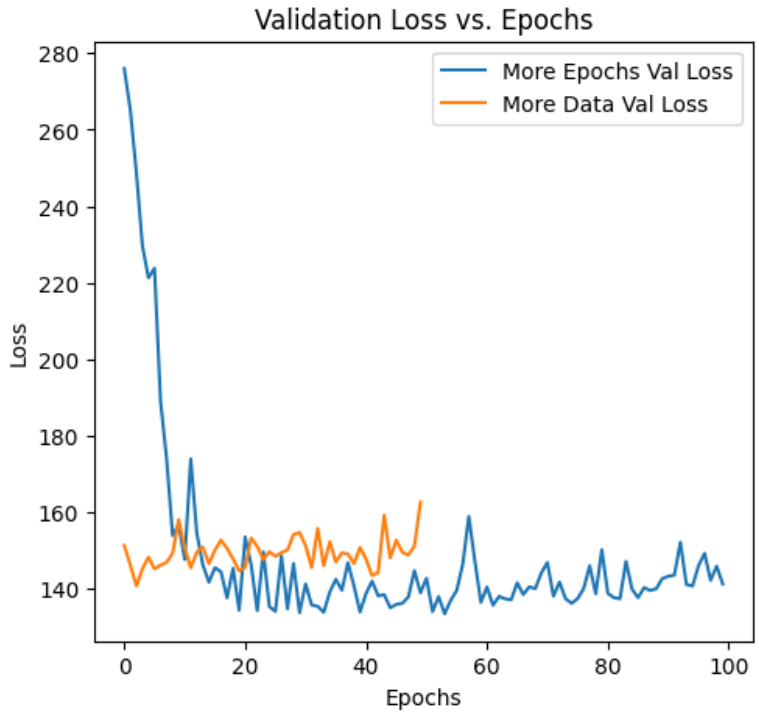


- Despite the changes, this approach didn't help at all.
- In fact, the validation loss actually got worse.

Test-3 (More Epochs or More Data)

- We sample half the dataset and train it for twice the epochs.
- sampling is done uniformly





- We can conclude that training lesser data for more epochs would be a better approach.

Final Training: Dataset Preparation

- We combine all out observations till now for the final training.
- Start by loading and concatenating the entire training dataset:
  - Total entries: 5.9 million
  - Randomly sampled half: 2.9 million entries
- We also remove outliers, which were a very small percentage of the total rows.
- Then we calculated the *wtdb* and *wtdw* attributes for each entry.

Final Training: Encoding Strategy

- Used the following encoding scheme:

$$\text{tonum3d} = \begin{cases} r : -[0,8,0,8,0,8,0,8] \\ n : -[2.5,2.5,2.5,2.5,2.5,2.5,2.5,2.5] \\ b : -[8,0,8,0,8,0,8,0] \\ q : -[8,8,8,8,8,8,8,8] \\ k : -[1,1,1,1,1,1,1,1] \\ p : -[0,0,0,0,0,1,0,0] \\ R : [0,8,0,8,0,8,0,8] \\ N : [2.5,2.5,2.5,2.5,2.5,2.5,2.5,2.5] \\ B : [8,0,8,0,8,0,8,0] \\ Q : [8,8,8,8,8,8,8,8] \\ K : [1,1,1,1,1,1,1,1] \\ P : [0,0,0,0,0,1,0,0] \end{cases}$$

Final Training: Fitting and Results

- Trained the model for 377 epochs (wanted to train for 500 however the kernel crashed because we ran out of credits)
- Final loss (Mean Absolute Error) obtained on the validation set was 152.63463 which is in stark contrast to the loss of 6012.80127 which we obtained with a very bare-bones NN model of (64 256 128 64 1)
- we also create a simple user interface for others to tryout our model @ mtl782-chessapp.streamlit.app
- While playing with our interface we noticed a relatively good fit with a margin of error, and when we looked for the cause we found that our model has a slight bias towards black as well as a tendency to compress evaluations around 0, as can be seen in the following histograms.

Final Training: