CS 254 Machine Learning
Project Final Report Format
Jackson Hall and Jakob Walburger

1. Introduction

We are interested in predicting the expected centipawn loss of a chess player's move given the current board and features of that board given to us by Stockfish.

Stockfish is one of the leading chess engines that uses a highly optimized tree search algorithm combined with a static evaluation function to score each potential future position. Positions are given a positive or negative score indicating whether white or black has the advantage, or zero if the position is a draw. The magnitude of this number is the position's pawn-score, defined such that a score of +1 means that white is better by the equivalent of one pawn, and –1 would mean the same for black. The score may have decimals too, where positional advantages such as space, king safety, pawn structure, piece coordination, initiative, and other factors are weighed to contribute to the overall score in a way that has been finely tuned by human grandmasters to create a strong static evaluation function.

When a player makes a move, they either play the best move according to the engine or a suboptimal one. Thus, for the maximizing player (always white), the pawn-score either stays the same or drops, and vice-versa for the minimizing player (always black), except in rare cases where the move played is optimal but was not found by the engine. If it is white on turn where the most accurate move scores +1.4, but the position after the move played is +0.8, they have lost the equivalent of 0.6 in their pawn-score. Ideally, the loss in score each move is low, so it is usually measured in centipawns, or tenths of a pawn, so white's centipawn loss (CPL) would be 6 for this move.

The general quality of play in a game can be measured by averaging the CPL over all moves for both players, and in general the side with the lower average CPL wins the game. Analyzing the patterns and instances where players have the highest CPL is therefore useful to identify areas where players can improve at chess, especially if this could be enhanced through machine learning on a large dataset.

2. Problem Definition and Algorithm

2.1 Task Definition

We are looking to predict the centipawn loss score/centipawn score and determine if the player is winning, losing, or if the game is a draw. We have used 2 different sets of features, the first which contains a list of numeric values given to us by Stockfish that represent the game state for things like space, king safety, etc. The second feature set that we have created is a binary board representation which will have a feature space of 8x8x12 which is all 64 spaces on a chess board along with the 12 different piece types. The value will have a 1 if that piece is present on that board space or a 0 if it is not. For the output we have created 2 different datasets so we can try different things. The first is the predicted centipawn loss value which came from

the Stockfish evaluation, this is just a single number. When we go and create our training/testing data this value is converted into a binary class of 0 if the CPL is less than 30 or 1 if the CPL is great than 30. Our second output will be a 1x3 space vector that will represent if player is winning, losing, or at a draw, this is a one-hot encoded representation of the data. The winning, losing, or draw labels are given to a set of features if the predicted centipawn loss was in a certain range of numbers, this all comes from chess theory.

Predicting centipawn loss on a particular move, especially given the user's rating, would be interesting in that it could be used to identify types of tactical motifs that lower rated players consistently miss while higher rated players normally play correctly. We could test the accuracy of the prediction by using Lichess.org's database of chess puzzles and finding the decision boundary for when a player is 50% likely to find the full series of best moves. This number should correlate with the strength of the puzzle as recorded in the database, which is a number calculated by Lichess for each puzzle that basically takes the mean rating of players who solve it successfully.

2.2 Algorithm Definition

To solve this problem, we used a couple different techniques which we developed and other techniques which we researched. We started by using a Random Forest Regression (RF) algorithm to try and solve this problem. The reason behind trying to use the RF first was that this algorithm can handle a missing data and is usually a good step before investing more time into other methods to see if we predict any results with some level of accuracy. Along with the RF we also tried to use a Multilayer Perceptron Neural Network (MLP) to try and predict the centipawn loss score for the player. This network used 4 fully collected layers of 256 nodes each along with a dropout rate of 20%, batch normalization used between each of the 4 layers, along with using the ReLU function for them all. A MLP model is a good choice for this as it can handle regression task well, since we didn't get great results from the RF model and a MLP is very powerful we were hoping that we would get some better results from the start.

There were 2 other techniques that we researched and tried out. Rather than trying to use regression to predict these centipawn loss values we found that a much easier task is to try and classify if the player is winning, losing, or in a draw. First, we decided to try and use a similar MLP network to one which was found in the paper "Learning to Evaluate Chess Positions with Deep Neural Networks and Limited Lookahead". They used similar data to ours and all we needed to do way to tweak our data augmentation so that we have the binary 8x8x12 representation of the board state as well using the centipawn score rather than the centipawn loss score and converting those values into a winning, losing, or draw state. This network consisted of 3 layers of 2048, 500, and 50 nodes respectively, along with batch normalization, a dropout rate of 20%, and they all used the ReLU activation function. The other technique which we tried used a convolutional neural network (CNN). This network used 2 2D convolutional layers which used 20 5x5 filters first and then 50 3x3 filters second. Each used the ReLU activation function along with batch normalization and a dropout rate of 30% after each layer. The end of the network used a fully connected layer of 500 nodes and then a softmax layer for the output.

3. Experimental Evaluation

3.1 Methodology

What are criteria you are using to evaluate your method? What specific hypotheses does your experiment test? Describe the experimental methodology that you used, and why do you think it is the right way to measure the performance. What is the training/test data that was used, and why is it realistic or interesting? Exactly what performance data did you collect and how are you presenting and analyzing it? Comparisons to competing methods that address the same problem are sometimes useful.

For the first 2 models we are using the dataset that contains the centipawn loss score as the target. This dataset is split into a 80/20 split and then the 20% that is left for testing is then split into testing and validation at a split of 50/50. The data which is being used now is the numeric representation of the board along with the binary classifications for centipawn loss. To evaluate the RF we just used the sklearn R2 score and accuracy score to compare the testing predictions and their actually values. For the MPL model we looked at the training loss and accuracy, as well as the validation loss and accuracy.

For the other 2 models we used a similar 80/20 split for the training and testing as well as splitting the testing data into a validation set as we did before. We now have a binary representation of the board along with our one-hot encoded classes for the board state. To evaluate these algorithms we used the training loss and accuracy as well as the validation loss and accuracy as we did before with our first MLP model.

3.2 Results

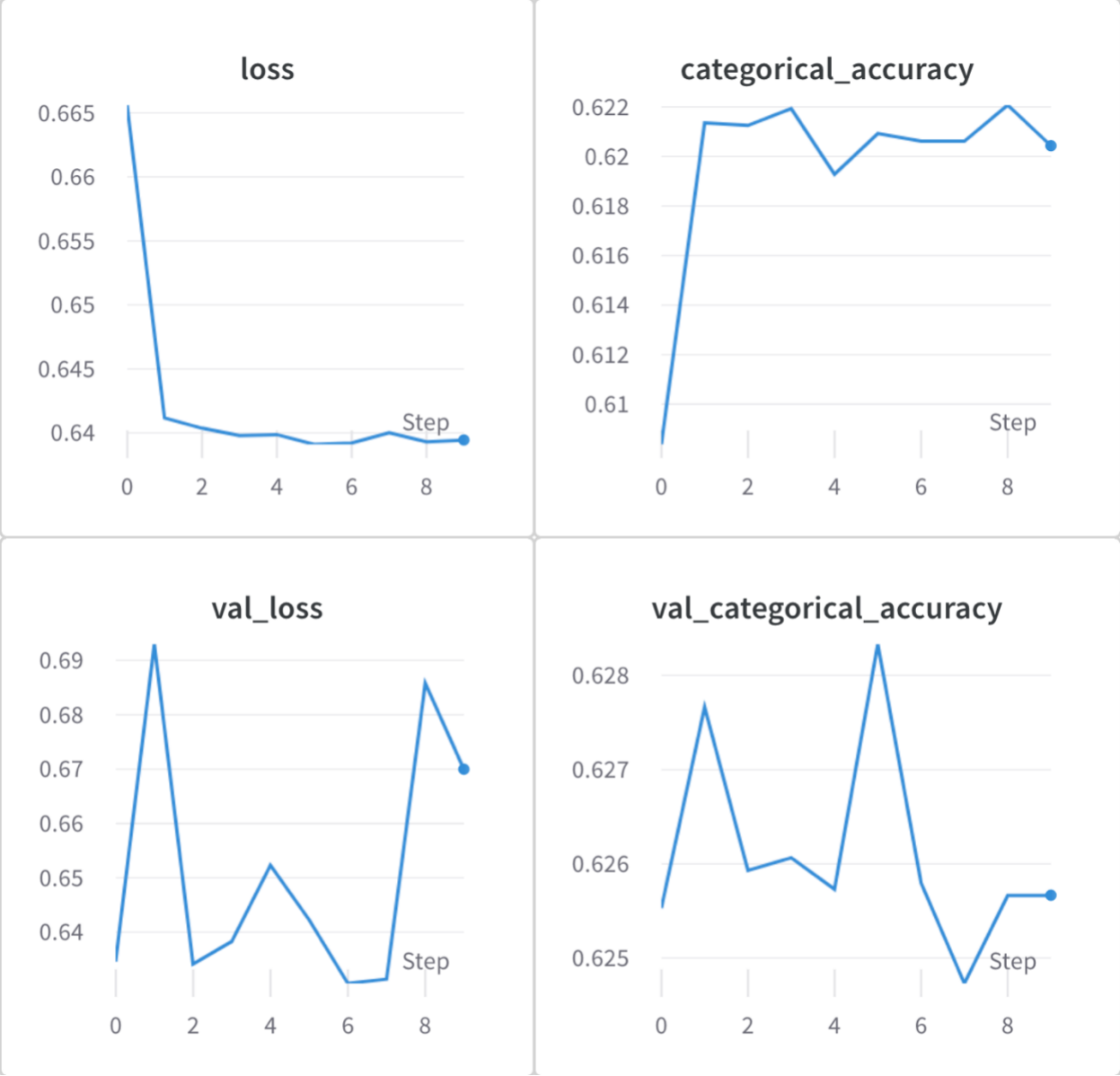
Present the quantitative results of your experiments. Graphical data presentation such as graphs and histograms are frequently better than tables. Show results examples, it would be great if you can show a demo.

Random Forest:

```
r2_score: -0.5099848374663145
Accuracy: 0.6452
```

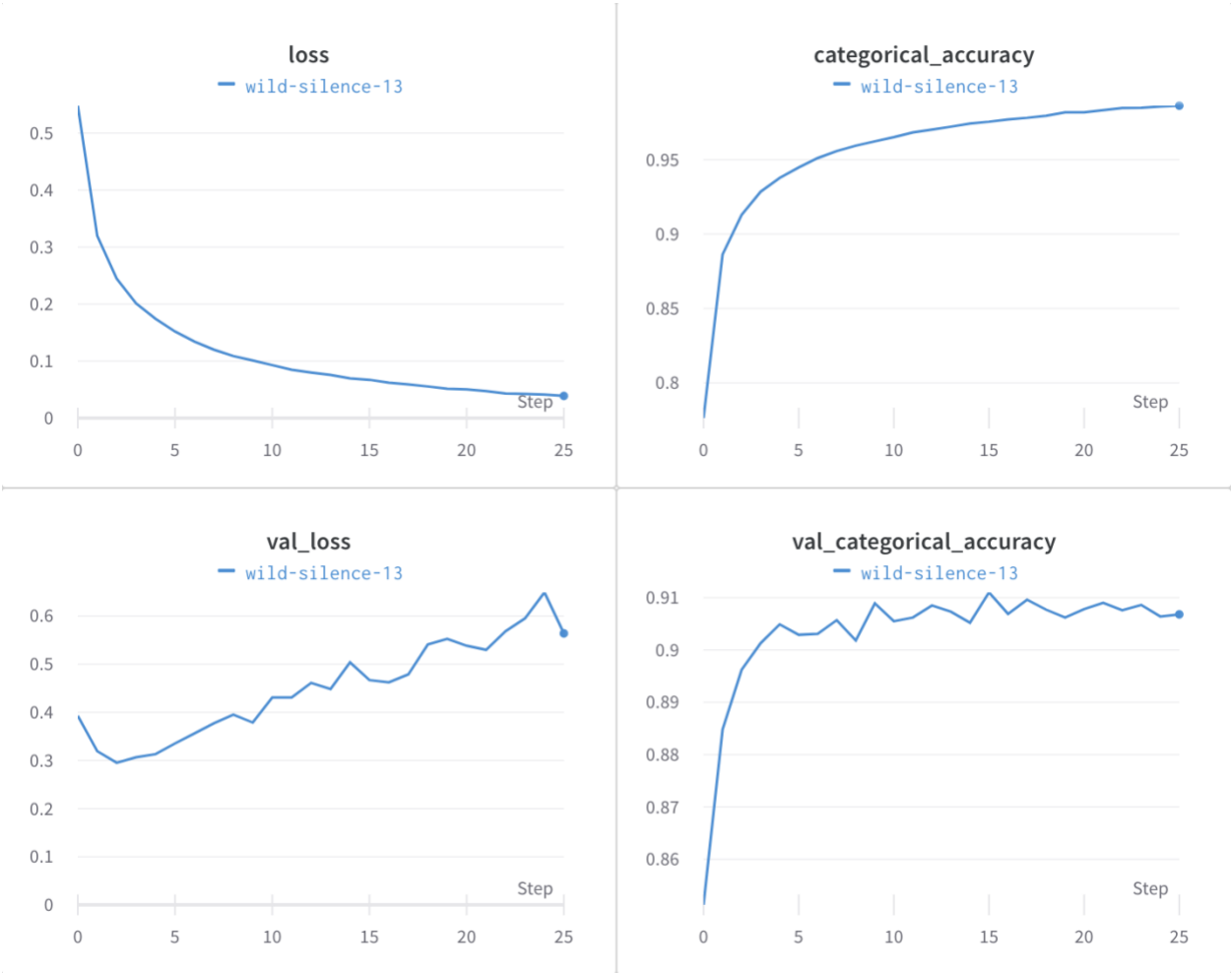
R2 and accuracy for the random forest. These are just to get a baseline to see if our other algorithms are performing well or not.

MLP 1:



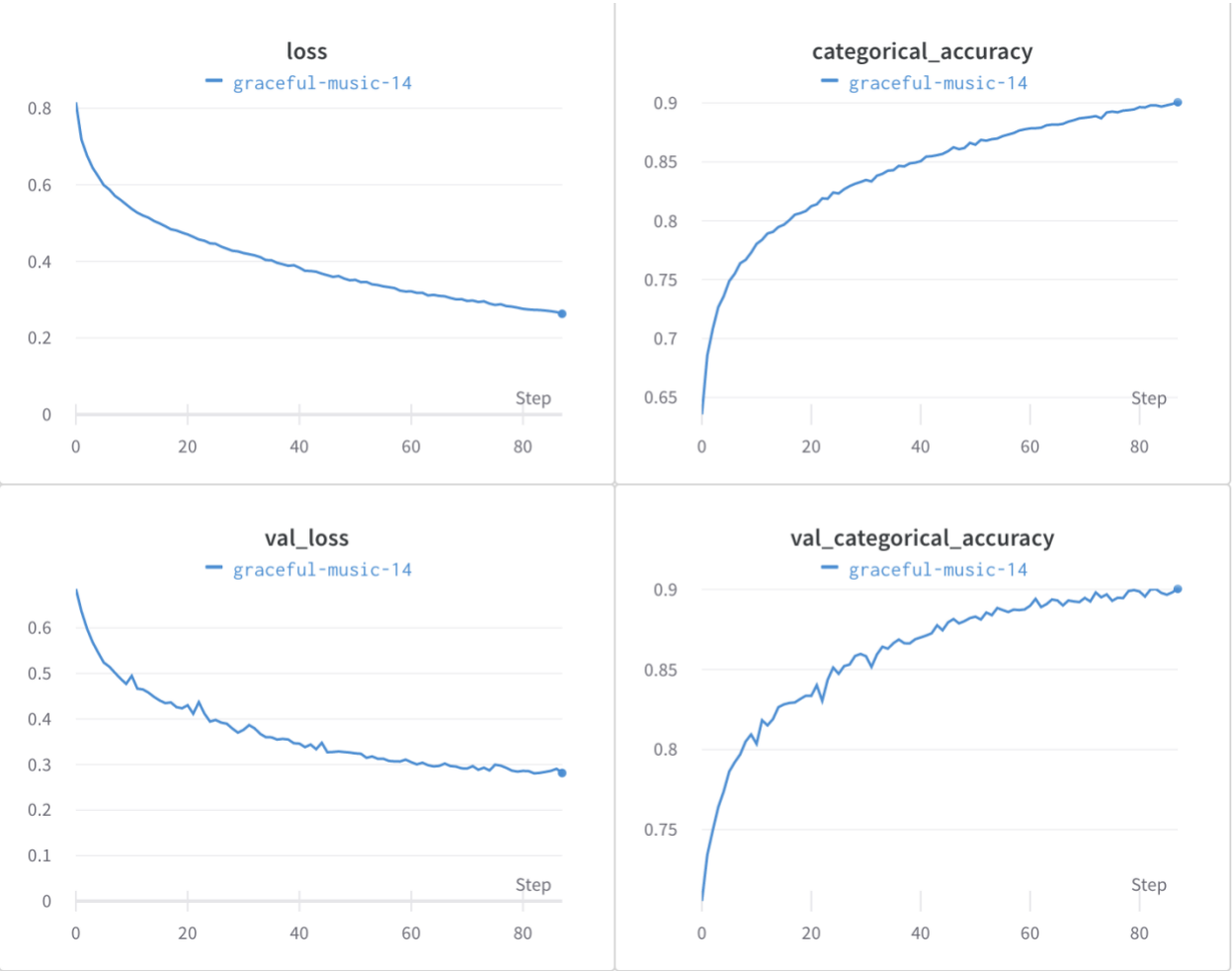
Accuracy around 62%.

MLP 2:



Accuracy around 91%

CNN:



Accuracy around 90%

3.3 Discussion

We can see that the 2 models we made from scratch are not performing very well compared to the researched models, this is to be expected. Reasoning behind this is that because we are trying to perform regression in the first 2 models to see if a centipawn loss score results in a change in the board state rather than try to predict the actual board state, which is a much easier problem, and allows us to draw the same conclusions as if were looking at the predicted centipawn loss values to determine the state. Overall, the CNN model performed the best. Even though the accuracy isn't as high as the second MLP model the loss and accuracy lines for training and validation data are much closer to each other than the other models. This suggests that it is generalizing well.

4. Related Work

Maia chess is another interesting project that has pre-trained several neural network chess engines on games stratified by their ELO rating. For example, they have a network trained on games played by players rated 1000-1200, 1200-1400, 1400-1600, etc. The 1200-1400 rated network is optimized to predict moves played by players in this rating range, for example, and is better at predicting moves for players in this class than any other of the networks. The appropriately rated network can further be trained on an individual's collection of games to better predict the moves of that player (McIlroy-Young et al., 2020). This indirectly accomplishes our same goal but is more complex than we hope to make the scope of this project.

Another piece of related work that we found and used heavily was a paper titled "Learning to Evaluate Chess Positions with Deep Neural Networks and Limited Lookahead". This paper was trying to solve a problem in a very similar fashion to our using almost the same data representation that we are using. They have created 4 different datasets and have used 2 different machine learning algorithms for each of the datasets. This paper was a large inspiration for us, and we used many of the same approaches using our datasets that we have created. Overall, this paper goes much further in depth about each of their algorithms, uses much more data, and is more accurate than the methods we have developed.

5. Code and Dataset

The GitHub repo can be cloned and run in a jupyter notebook. All that is needed is to run the file. You might run into some file path configurations that need to be changed, this is all dependent on the user's computer.

https://github.com/jdwalbur/CS254MLFinalProject

https://database.lichess.org/

https://stockfishchess.org/

6. Conclusion

Overall, we have accomplished what we set out to do. We wanted to try and develop a method to evaluate the chess board so we can make an informed decision about the current game and the advantages of one move compared to another. By using many different methods along with different representations of the data we have covered many bases for ways to approach this problem. The CNN is working best for us based on the validation results compared to the training and testing results. In the future it would we can continue the development of the CNN to try and predict the centipawn loss score rather than the board state. This would be very helpful in comparing different moves of a certain board.

Bibliography

Lichess.org Open Database. (n.d.). Retrieved October 3, 2021, from https://database.lichess.org/

McIlroy-Young, R., Sen, S., Kleinberg, J., & Anderson, A. (2020). Aligning Superhuman AI with Human Behavior: Chess as a Model System. 10.1145/3394486.3403219

Stockfish Evaluation Guide. (n.d.). Retrieved October 3, 2021, from https://hxim.github.io/Stockfish-Evaluation-Guide/

Chess: A chess library for python¶. python. (n.d.). Retrieved November 1, 2021, from https://python-chess.readthedocs.io/en/latest/.

Wikimedia Foundation. (2021, May 14). Forsyth–edwards notation. Wikipedia. Retrieved November 1, 2021, from https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation.

Sabatelli, Matthia, et al. Learning to Evaluate Chess Positions with Deep Neural Networks and Limited Lookahead. https://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/ICPRAM_CHESS_DNN_2018.pdf.