A comparison of generalization techniques on supervised chess evaluation functions

WP Swart

orcid.org/ 0000-0003-3674-6560

Dissertation accepted in fulfilment of the requirements for the degree Master of Science in Engineering Science with Computer and Electronic Engineering at the

North-West University

Supervisor:	Prof PA van Vuuren
Co-supervisor:	AJ Alberts

Graduation:	June 2021
Student number:	24159565

Declaration

I, Willem Petrus Swart hereby declare that the dissertation entitled "A comparison of generalization techniques on supervised chess evaluation functions" is my own original work and has not already been submitted to any other university or institution for examination.

W.P. Swart

Student number: 24159565

Signed on the 9th day of December 2020 at Johannesburg.

Acknowledgements

Prof Pieter, without you this would never have happened.

Andreas, dankie vir jou (in volgorde van dankbaarheid) raad, geselsies en lang rekenaar.

Wian and Arnold, thanks for (at least attempting) putting up with me.

If you're still reading these, consider yourself acknowledged.

Abstract

This dissertation was created in pursuit of comparing different deep neural network generalization techniques on neural networks that were trained by supervised optimisation to be used as chess evaluation functions. Investigations were performed on training material, network anatomy and on specialized generalization techniques. These experiments were evaluated firstly as normal classifiers, and then as part of a fully fledged chess agent. It was found that deep convolutional neural networks perform exceptionally well in both of the criteria mentioned, but that their hyperparameters may be harder to optimize. A strong correlation between classifier performance and chess prowess was also noted, suggesting that classification metrics can in a broad sense be used as a gauge of chess performance.

Keywords: *Chess, Deep Neural Networks, Supervised Learning, Generalization*

Contents

Appendices

List of Figures

List of Tables

Chapter 1

Introduction

This chapter describes the motivation to the study of generalization techniques on a supervised neural network Chess evaluation function. The background to the research is presented before a literature review on the subject is given. A research question is formulated and then an experiment to answer said question is defined.

1.1 Background

On the 3rd of May, 1997, Deep Blue became the first machine to defeat a reigning world champion in a series of chess matches. Deep Blue, a super computer designed specifically to play chess, introduced an era where Minimax [1] chess engines would dominate the game for more than 20 years [1]. That all changed when Alphabet Inc's DeepMind team decided that chess was the next game that they would unleash their deep neural networks on.

In 2015, DeepMind's AlphaGo became the first computer program to beat a professional Go player [2]. Soon after, AlphaGo Zero was developed to learn Go only from self play (zero prior knowledge, hence the "Zero" moniker). In 2017 AlphaGo was revealed as a more general en-

[1] Minimax is a decision rule for minimizing the possible loss for a worst case scenario. In a perfect information game, it assumes that the opposing player will play optimally (the worst case), and aims to **mini**mize loss when the opposing player's gain is at its **max**imum.

gine, that could defeat world class opponents in Go, Chess and Shogi [3][2]. In a 1000 game chess match, AlphaZero defeated one of the most successful chess engines, StockFish [4], with a final score of 155-6[3]. This was the dawn of a new era in computer chess. Since then, neural network based chess engines have ascended the rankings of the Top Chess Engine Competition (TCEC) [5] , with Leela Chess Zero [6] losing the Season 14 Super Final with only half a point, and winning the Season 15 final with 4 points [7]. The season 16 final was contested by Stockfish and a new contender, AllieStein.

AlieStein is a neural network based chess engine that uses an evaluation function that is trained using previous games with supervised learning. The authors of Allie cited this choice as an attempt to reduce training time, claiming that they could retrain their entire network within weeks as opposed to Leela, which would take months [8]. Another contender that is willing to wander from the path trodden by the self-play flock is Fat Fritz [9]. Fat Fritz uses a combination of supervised learning techniques and reinforcement learning to train its neural networks, and has shown favourable performance against king of the hill, Stockfish [10]. These developments begs the question of what can be done to improve the supervised evaluation functions incorporated in these new engines.

Chess evaluation functions have been thought about since at least the era of Shannon [11]. For chess, an evaluation function is a function that evaluates a board position as advantageous for a specific player (a disadvantage being implied for the opponent). Simple evaluations can be performed by assigning pieces values relating to that piece's power, and summing over the pieces on each side to obtain a score that can predict who has the advantage in a match. A similar, though far more intricate, process is what was used to beat Gary Kasparov at the end of the 20$^{\text{th}}$ century. This approach has reigned supreme until recently, but now neural network based chess engines are considered, by some, at least as good as the engines employing man-made evaluation functions.

Using rigorous mathematics, it can be shown that a function exists, that when presented with any given state of a chess match, it will return the eventual victor, given that at least one player plays perfectly [11][4]. It can also be shown that a multilayer perceptron (MLP) is a universal

[2]It is worth noting this engine can only play the game that it was trained for, but that it has the ability to train for the mentioned games.

[3]The other 839 games resulted in a draw.

[4]This is due to the fact that the amount of possible positions, while enormous, remain finite.

approximator [12] of any continuous function, with as little as one hidden layer. This makes it easier to believe that neural network based chess engines are outperforming the more traditional heuristic evaluation based engines, since they have more potential – in theory at least.

In practice, however, there are a few caveats. To approximate the aforementioned function, using supervised learning, some training samples are needed. This is problematic because it requires one to know what the eventual outcome of some chess positions are, and to be certain of said outcomes, the optimal strategy needs to be known. If the optimal strategy was known, then an approximation of the evaluation function would be redundant.

The next best strategy is to try and approximate perfect play by sampling from the **billions** of chess matches that are available in online archives. Any supervised model is only as good as the data it is trained on, and with the amounts of data available, it is not hard to believe that an evaluation function obtained through such a manner, can compete with the best man made functions with the same goal.

Generalization in neural networks refers to the ability of a network to notice underlying features in data, instead of just memorizing training data. If neural network chess evaluation functions can notice the advantageous patterns of winning chess positions, they might just receive the nudge they need to undoubtedly surpass their man-made counterparts.

1.2 Literature Review

This section aims to illustrate previous attempts at neural network based chess engines, how they were implemented and how they fared.

1.2.1 On neural network based chess engines

NeuroChess [13] evaluated board positions with a neural network that used 175 handcrafted features as input. Temporal-difference learning was used to predict the winner of a game, and also the expected state after two moves. NeuroChess won 13% of games against the min-max based GnuChess using a fixed depth 2 search, but didn't win the match.

KnightCap [14] evaluated positions by a neural network that used an attack table - a table that listed which pieces attacked what, and which pieces were defended by which. A variant

of Temporal-difference learning, known as TD(leaf), was used to create the model. TD(Leaf) updates the leaf value of the principal variation of an alpha-beta search. KnightCap achieved human master levels of play (Elo[5] rating of 2200) after training against a computer opponent with selected initial material weights.

Meep [15] evaluated positions by a linear evaluation function based on handcrafted features. It was trained by another variant of temporal-difference learning, known as TreeStrap, that updates all nodes of an alpha-beta search. Meep defeated human international master players in 13 out of 15 games, after training by self-play with randomly initialized weights.

Giraffe [16] evaluated positions by a neural network that included mobility maps and attack and defends maps describing the lowest valued attacker and defender of each square. It was trained by self-play using TD(leaf), also reaching a standard of play comparable to international masters.

DeepChess [17] trained a neural network to perform pair-wise evaluations of positions. It was trained by supervised learning from a database of human expert games that was pre-filtered to avoid capture moves and drawn games. DeepChess reached a strong grandmaster (Elo 2500) level of play.

AlphaZero is a deep network based chess engine that trained entirely through self play. It defeated the 2016 Chess Engine World Champion Stockfish in an exhibition match. It has an Elo rating of approximately 3300, while the highest rating ever achieved by a human was by Magnus Carlson at 2882 [18].

LeelaChessZero is a neural network based chess engine that attempts to mimic the work done by Google's AlphaZero, but in an open source, distributed computing way [6].

1.2.2 On generalization in neural networks

Generalization in neural networks refers to the ability of a model to function correctly even if there are differences between the data that is used to train it, and the data used for inference. A good example of this so called generalization can be observed in something like hand written digit recognition. The MNIST dataset consists of 70'000 images of handwritten digits, but it

[5]The Elo system, after creator Arpad Elo, is a rating system used to predict the performance of opponents in games where a finite number of results are possible. Three in the case of chess.

is likely that nor the writer or the reader's digits feature in this dataset. It is more likely still, that even a rudimentary neural-network trained on MNIST will be able to recognize a two or a three, created by either of us. This is due to the network's ability to generalize.

Understanding why neural networks generalize so well has been an open question for some time. It is not uncommon to train a network with a large amount of parameters on training samples that number an order of magnitude less (10 times more parameters to train than samples to train them on), and expect good results. Work by Wu et al. [19] show that the characteristics of the loss landscape of neural networks and the fact that networks are initialized randomly causes a converging classifier to be highly probable.

As is very much expected, we are not content with the current state of generalization in neural networks, and naturally there have been attempts to improve the state of the art. In a paper published at the end of 2017, Keskar et al. stated that although adaptive optimizer like Adam and RMSprop converge faster that traditional gradient descent methods, they tend to generalize worse. They obtained promosing results by employing what they called: "SWATS", or the **Sw**itches from **A**dam **t**o **S**GD method. This method, like the name implies, starts by optimizing the model using the Adam optimizer, and after a threshold training accuracy is reached, it switches to SGD (Stochastic Gradient Descent).

Data augmentation is another method used to improve generalization. It follows from the idea that more data, means more variance, which in turn will prepare a model for a larger number of cases. Work done by Zhou et al. have shown that while batch normalization is great tool for improving generalization [20], in the presence of augmented data, there is not much to gain. They therefore proposed a novel technique, dubbed Consistent Normalization, with favourable results in image classification and segmentation.

1.3 Research Question

The main question that this research attempts to answer is:

Can a chess evaluation function, obtained by supervised learning, improve the chess performance of the chess engine it is part of, by using generalization techniques that will result in a

better classifier[6]?

In other words, is the relationship between a chess evaluation function's ability to predict which player will win a chess game and the chess prowess of the engine that employs it strong enough that increasing the classifier's ability will improve the engine's performance?

Since machine learning depends mainly on the data, the model, and the way the model is made to fit the data, various approaches will be investigated in an attempt to increase chess capability.

These approaches will be investigated to answer the following sub-questions:

- Will adjustments to the representation of the data improve the performance of the evaluation function?
- Will adjustments to the neural network anatomy and topology [7] improve the performance of the evaluation function?
- Will adjustments to the training phase of the model improve the performance of the evaluation function?

At this point the reader should note that a better chess engine is considered a side-effect of the research, and that the main goal was to investigate generalization in neural networks. Chess is used as a special real world benchmark of the generalization of the neural networks in question.

[6] A classifier in this context refers to a system that will group similar chess positions and assign them to a class, like winning

[7] Throughout this dissertation, anatomy will refer to the innards of the neural network. The difference between a multi-layer perceptron and a deep convolutional network is due to anatomy. Topology will refer to how the innards of the network is arranged. The difference between a deep neural network and a shallow one is due to topology

1.4 Method

This section details the basic setup of the experiments, the metrics used to asses the results and the method of validation adhered to.

1.4.1 Experimental Setup

To evaluate the effect of these techniques, a baseline model was needed. Ideally, the neural networks used by the state of the art chess engines should be used, as they are tried and tested, but unfortunately these networks are trained on hardware that is out of the reach of most mere mortals (At the time of writing, it was estimated that AlphaZero would require 1800 years of GPU time on a single top of the range consumer GPU [21]), so another baseline needed to be developed. This baseline was calibrated against a state of the art engine, Stockfish. The development and validation of this baseline is documented in chapter 2.

Every technique was investigated by applying said technique to the baseline, retraining the model and evaluating the model as both a classifier and a chess evaluation function. To evaluate the performance of the model as a classifier, the successful prediction of the ultimate state of the game from any given position is measured. The expertise of the model as a chess evaluation function will be assessed by using the model as part of a chess engine in real world tournament scenarios.

Individual experiments are detailed in the following chapters, but an overarching setup is given here.

In all cases except those where explicitly stated otherwise, the model was developed on approximately 500,000 chess games. There are far more available but constraints on time and money limited the writer to the amount stated. Training and tests sets were constructed by randomly sampling games with a distribution of 88:12 for training and testing respectively. These numbers were chosen due to the way the data training data was munged, and was an attempt to get as close to an 90:10 ratio, which is considered sufficient whenever the dataset gets large.

In all cases except those where explicitly stated otherwise, three different seeds of each model were trained, each by training over five epochs of the data. The model was trained using the cross-entropy loss function, using the ADAdelta optimizer, using mini-batches of size 256.

1.4.2 Validation

To gauge the effect of the techniques employed, the model was evaluated in a chess engine, playing chess matches, and as a classifier.

Chess performance will be measured by deploying each seed of each version in a chess engine, and playing 500 chess games against Stockfish at reduced ability (explained in chapter 2), and then calculating an Elo rating for each seed, and for each version.

Classification accuracy will be assessed against :

- Training accuracy
- Validation accuracy
- Normalized validation accuracy
- Use case accuracy
- Normalized use case accuracy
- Class accuracy

Training accuracy refers to the accuracy with which the model can classify data on which it has been trained. Validation accuracy refers to accuracy on data which is completely unknown. In both cases of normalization, it refers to adjusting the accuracies to represent how they would've looked if the class distribution was uniform (The training data distribution was skewd by the fact that white did not win, draw or lose regularly in any data set). More details are provided in chapter 2, but it entails comparing the difference between the predicted probability of winning and losing, to the eventual outcome of a game. Class accuracy refers to the classifier's accuracy pertaining to data in a specific class (White winning, white losing, a tie).

1.5 Subsequent Structure

The remainder of this dissertation contains the following chapters:

2 - **The construction and calibration of the baseline**

In this chapter, the design of a baseline evaluation function is detailed. Afterwards, the process

of integrating said function in machinery to play chess is illustrated before the entire system is benchmarked to measure not only evaluation performance, but real-world chess capabilities.

3 - **Manual input encodings, embeddings and datasets**

In this chapter, the effect of the format of the data presented to the model is investigated. Manual encodings, trained embeddings and smaller datasets are applied and the impact on classification performance and chess capability is ascertained.

4 - **Network anatomy and topology**

In this chapter, the effect of the network anatomy (the types of ”neurons” used in the neural network) and topology (the depth and width of the network) on the chess evaluation function is investigated. Convolution kernels are compared to dense layers, while various width and depth configurations are evaluated for both. This is all in a quest to evaluate chess capabilities and classifier performance.

5 - **Balancing under- and overfitting**

This chapter will document the investigation of techniques generally used to reduce overfitting. Overfitting is the largest detractor of a general neural network, which means that reducing overfitting will increase generalization. These techniques will be applied to the baseline neural network, and will be evaluated in terms of chess performance and classifier accuracy.

6 - **Conclusion**

Chapter 2

The construction and calibration of the baseline

In this chapter, the design of a baseline evaluation function is detailed. Afterwards, the process of integrating said function in machinery to play chess is illustrated before the entire system is benchmarked to measure not only evaluation performance, but real-world chess capabilities.

This dissertation assumes a (slightly more than) basic knowledge on neural networks. For a great reference, Chapter 5 of the book, Pattern Recognition and Machine Learning, by Chris Bishop [22], should be consulted. For a broader overview, Deep Learning, by Ian Goodfellow [23], is an excellent resource as well.

2.1 Obtaining a baseline evaluation function

As mentioned in Chapter 1, the ideal evaluation function will map any chess position to the eventual winner of the specific game. In practice, if this function is not perfect but it behaves as if it is, using it in a chess engine will result in poor play. Therefore chess engines use evaluation functions that need to map specific chess positions to some metric that indicates which player is leading, and by how much. Such an evaluation function will now be developed.

The first step is to determine the domain of the evaluation function by understanding the range of a chess position. Any chess position can be completely described using a (Forsyth Edwards Notation) FEN-string. The following FEN-string describes the starting position:

"rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1".

This string can be divided into three parts. The first ("rnbqkbnr/ pppppppp/ 8/8/8/8/ PPPPPPPP/ RNBQKBNR") part is the part describing how the pieces reside on the board. With a chessboard in front of a player, and the first rank nearest to them, start with the square at the far left. Move from left to right, and down. Each letter represents a piece, with the letter used to start the piece's name signify that letter (r for rook, q for queen) except for the knight, which is signified by an n. Lower case letters represent black pieces and upper case the white pieces. Numbers represent empty squares along a rank. In the example above, the 8's signify an empty rank.

In the case that white opens with e4, the string would look as follows:

"rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 0 1".

Notice the "4P3" part. It means that on the 4^{th} rank there are four empty squares, a white pawn, and three more empty squares.

After the position, the next important part is the side to move. In the first string, this was a 'w', showing that white is to move. Then white moved, and in the second string, the 'b' showed that it was black's turn.

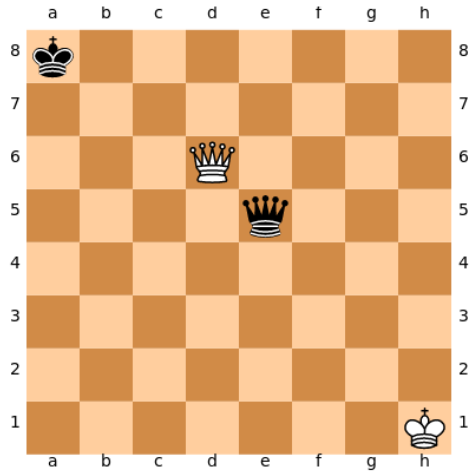

Figure 2.1: Side to move example

The next part shows castling rights. The letter k and q means king- and queenside respectively, and the typecase of the letters once again signifies the color. "KQkq" means that both the white and the black king can castle to both sides, whilst "Qk" would show that the white king can castle queen's side, and the black king can castle king's side.

The final part signifies the *en passant* squares, which are the squares on which an *en passant* capture can be made. The remaining numbers represent a half-move clock, used to determine draws and a full move

counter, counting the number of moves of the match.

It is at this stage worth noting that the first part describes the physical position, and the rest describe legal moves. Since move generation is typically not related to position evaluation, all but one of the legal move parameters can be ignored.

The side to move cannot be ignored, and Figure 2.1 illustrates why. In the figure, the board is symmetrical with respect to material, but as soon as the side to move is known, the other side is at a significant disadvantage. Therefore the side to move needs to be encoded in the input data.

An elegant way to do this is to assume that white will always move next, and when a position with black as the mover is presented, to mirror the board and reverse the colours. This process is illustrated in Figures 2.2 and 2.3.

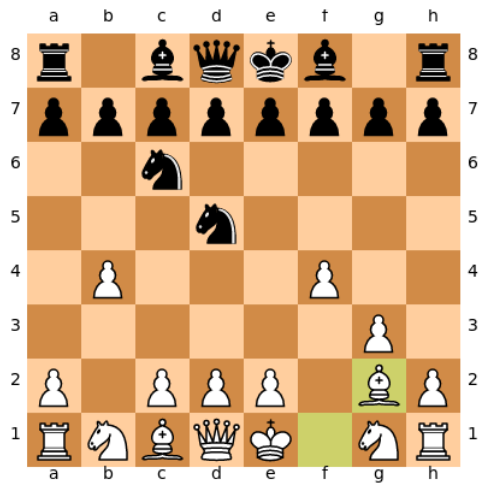

Figure 2.2: Standard position

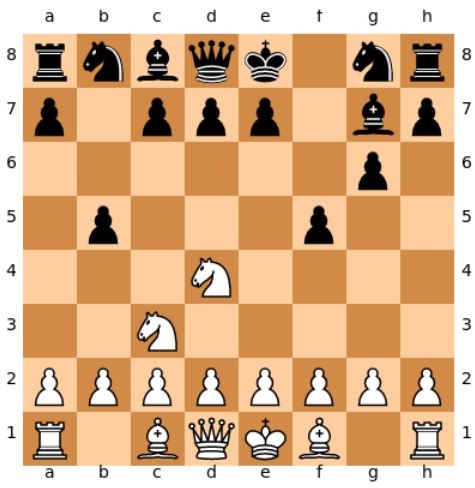

Figure 2.3: Mirrored Position

In Figure 2.2, white moved it's bishop from f1 to g2. It is therefore black's turn to move. In Figure 2.3 the exact same position is represented, except that it is now white's turn to move, eliminating the need to encode the side to move in the data.

Now that move legality has been suitably addressed, the pieces on the board can be represented. There are 12 different[1] pieces that need to be encoded on 64 different squares. To avoid assigning significant numerical values to pieces, one-hot encoding will be employed to

[1]The six pieces, for both colours

acquire a representation. (One-hot encoding represents categorical data with vectors instead of numbers. For example, the classes red,green,blue, can be represented as [001], [010], and [100], instead of 1,2 and 3.) This has the implication that 768 bits are needed to encode the position. A 64 bit mask that represents the 8x8 chess board times 12 to represent the different pieces.

A mapping from a 768 dimensional input to a three dimensional output (win, draw or lose for the player to move) is therefore needed. The MNIST[2] dataset consists of 28x28 images, which means that it has 784 input dimensions, and 10 output dimensions. It therefore stands to reason, that a mapping that can do a reduction from 784 to 10 dimensions can map 768 dimensions to 3.

The deep learning library Keras [24] was used to develop all the models in this research, and, because it showed promise, the architecture of the Keras MNIST example classifier [25] was used as a baseline evaluation function. This choice was initially made for convenience' sake, but as the results show, as a classifier, this configuration fared acceptably. The only modification was a reduction in nodes in the output layer, from ten for MNIST, to three for chess. This classifier achieved a 99.25% test accuracy on MNIST data. The architecture is displayed in Table 2.1.

Table 2.1: Architecture for control network

Layer (type)	Output Shape	# of Params	Activation
Conv2D	(12, 8, 32)	2336	ReLU
Conv2D	(12, 8, 32)	9248	ReLU
Conv2D	(12, 8, 32)	9248	ReLU
Max Pooling	(6, 4, 32)	0	—
Dropout	(6, 4, 32)	0	—-
Flatten	(768)	0	—
Dense	(64)	49216	ReLU
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable parameters Non-trainable parameters Total parameters		70243 0 70243	

[2]MNIST is a large collection of handwritten digits

2.2 Interfacing between the evaluation function and an engine

A static evaluation function is almost worthless by itself. For it to provide value it needs to be part of a chess engine. The *de facto* standards for chess engines are luckily both open source software. Stockfish [4] and Leela Chess Zero (stylized as lc0) [6] are both publically available on Github, under licensing that allows everyone to modify it. They do, however, follow a vastly different approach in playing chess. lc0 uses a Monte Carlo Tree Search approach to generate possible moves, while Stockfish uses a pruned exhaustive tree. Further, Stockfish uses a heuristic evaluation while lc0 has a neural network based function.

Stockfish was chosen to host the new evaluation function because of a feature that allows it to play at a calibrated skill level which is comparable to a human player at a specific Elo rating. This *calibrated* version of the engine achieves different levels of play by electing to play sub-optimal moves with a varying probability depending on the level of play that is expected. Stockfish is not the only engine that is capable of this feat, but this, in conjunction with the readily available and maintained source code, made Stockfish an easy choice.

Using Stockfish as a host is not without drawbacks, though. The most prominent one is that Stockfish's evaluation function is based on extremely fast bit-wise operations, and the rest of the engine is designed with this speed in mind. A single core 2.4GHz CPU can do about 1.5 million evaluations per second while the same CPU will, depending on the size of the network that is used, manage at least 100 times less feed-forward passes of a neural network. The rest of the engine is designed around this lightweight and fast evaluation, and just bootstrapping an evaluation function that takes orders of magnitude more time to complete leads to problems, like time related forfeits[3]. It is worth reiterating that this research is not about chess engines in particular, but limits its scope to the evaluation function, and it's evaluation performance in particular. To get a fair comparison the Stockfish evaluation function while being economical with time, the search depth was limited to N=50000 nodes of the each player's search tree.

Now, to successfully perform Minimax, the evaluation function needs to return a useful evaluation that either specifies which player is definitely going to win, or which player is ahead. The probability of the three outcomes yielded by the baseline evaluation function is not worth much on its own. Traditionally an evaluation returns the advantage a player has in a specific

[3] In a tournament chess, a player that has zero time left is forced to forfeit the game.

position in terms of centipawns[4]. The developers of lc0 uses the following equation to do a conversion between expected win percentage and centipawn advantage:

$$s = -650 \times \log\left(\frac{1}{x} - 1\right). \tag{2.1}$$

In equation 2.1, x denotes the expected outcome of a player to win. This relation is purely for comparison to traditional chess engines, and cannot be used as more than that, since it only takes into account the probability of a single player to win. Usually in chess, the advantage one player has is equal to the disadvantage the opposition has. This is not the case for the new baseline, however, as the third outcome, drawing, renders this zero sum supposition invalid. (As an example, the classifier can predict a 30% chance for white to win, a 5% chance for white to lose, and the remainder for a draw. The relation in equation 2.1 would lead an engine to believe that white is behind, while this is clearly not the case.)

A better approach would be to compare each probability individually. The difference between the chance for white to win and the chance for black to win gives a metric to asses which player is actually leading and was used throughout these experiments. A small problem arose as a result of the aforementioned metric, since internally Stockfish uses integer values for static evaluations. As a solution to this problem, this metric was simply multiplied by a large number (10000, to save as much information as practically possible), before converting it to an integer.

2.3 Calibration of the baseline

Before measuring the effect of a more general neural network on chess performance, it is useful to know how well the network performs from the get go. As the evaluation function in question is based on the idea of the perfect evaluation function, it is interesting to know how a traditional evaluation function fares when it needs to predict the eventual winner. The ideal evaluation function assumes perfect play, and although there is no way of knowing whether play was in fact perfect, a quantifiable value can still be attached to accuracy. It was also necessary to find out how well the new baseline actually played chess.

[4] A centipawn is the value of one pawn divided by 100.

2.3.1 The metrics used

The new evaluation function's performance was evaluated by considering how accurately it could predict the eventual outcome of a chess game. Since Stockfish's evaluation function yields a numerical value indicating which side is leading, it was used as proxy for the eventual winner. Also, the "use case" metric used to evaluate the neural network borrowed from the TCEC draw rule[5], so for Stockfish to correctly predict a tie, a value of

$$|eval| < 8Cp \tag{2.2}$$

will be used.

Similarly, the baseline evaluation function was compared to Stockfish in a series of 500 game chess matches. In compliance with the Universal Chess Interface (UCI) [26], Stockfish can play at a reduced skill level, to mimic human Elo ratings. The baseline competed against Stockfish at an artificial Elo rating of 1800, as the two engines were closely matched at this setting. This metric does not mean anything concrete, though. Stockfish was calibrated against the CCRL [27] (Computer Chess Rating List) 's 40/4 time control. The depth limit imposed on both engines means that the chess won't be at the level of, for example, an 1800 Elo rated chess engine. In practice, the baseline won't reach this level without the depth limit, since in the allotted time, it will reach a much shallower depth of the search tree (less positions can be evaluated). A shallower search causes complex transactions to be interrupted, which means a misleading result will be reported about the strength of the current position. The Elo ratings stated throughout the rest of this document serves as an indication to what should be possible, if the evaluations could happen as fast as originally intended.

2.3.2 Calibration results

Before any of the above can be established, there is something to be said about the core of every supervised learning exercise – the data. For the creation of the baseline two sources of data were considered. The TCEC archives [5] consists of 11106 usable games with about 1.4 million different positions and the Lichess [28] database, at the time of writing, consisted of about 1.1 billion games and (extrapolating violently) 66 billion positions. Lichess is an online chess playing service that allows any *agent*, be it human or otherwise, to play chess against any

[5]The draw rule comes into effect when both engines report a score of less than 0.08 pawns for 4 plies

opponent. These games are then saved in the aforementioned database.

In an attempt to mimic how most human chess players learn chess, the large dataset of human chess was used to learn the basics of chess, before the trained model's parameters were used in a transfer learning attempt to learn the finer nuances of *excellent* chess. In these experiments, a subset of 450000 games was sampled from the Lichess data. For completeness' sake, the model was trained on both the human (Lichess) and computer (TCEC) sets individually, before the transfer learning was applied. As a reminder, the accuracies reported in the following tables are split into traditional validation accuracy, and use case accuracy. For traditional accuracy, the output of the classifier was compared to the labels of the validation set, and for the use-case accuracy, the difference between the winning and losing probabilities were compared to the validation labels.

Before the results are displayed, a note about the structure of the tables containing them. These tables are wide tables with many columns that have been condensed into a convenient form factor. The first column, Seed, refers to a specific random seed that was used to train each network. The seed identifiers should be used to link the results of the same networks in different rows.

The rest of the columns should be interpreted as follows:

- Training Accuracy: The final accuracy that the model achieved during training, evaluated on samples that were used in training, and therefore not novel.
- Validation accuracy: The accuracy that the model achieved after training, evaluated on training samples that were not presented to it before.
- Normalized Validation Accuracy: The same validation accuracy as above, but normalized as if the distribution of the validation data was uniform.
- Use case accuracy: Validation accuracy, but measured against how the classifier will be used in a chess engine. For example, a classifier might output the following probabilities: 0.4, 0.4 ,0.2 for white winning drawing and losing respectively. In the operation of the engine, the winning side would be calculated as white, since the predicted probability of white winning is higher. If white did end up winning, this was counted as a true positive.

- Normalized UCA: The Use Case Accuracy described above, but once again normalized as if the evaluation samples were uniformly distributed across the output space.
- Win, draw, and lose accuracy: The evaluation accuracies of the classifier in each of the respective classes.
- Leading Accuracy and Losing Accuracy: Use case accuracies, but for each class. Drawing accuracy was not included since a ”drawing” prediction has no value in the use case, and because the window in which a draw was predicted was really small. See Chapter 2.1.
- Win, draw, and lose count: The accumulated results that each classifier had when playing against an opponent.
- Score: The difference between Win count and Lose count, important for the calculation of the Elo Rating.
- Elo Rating. The expected Elo rating of the classifier in question.

Table 2.2 shows the results when using the TCEC data for training. Table 2.3 show the results when training on the much larger Lichess dataset, and Table 2.4 shows the results when training on the Lichess data first, before training on the TCEC games.

In all three versions of the experiment, the model was trained over five epochs of the data using the ADADelta optimizer, with mini-batch sized of 256. The model was evaluated against another sample of the Lichess database. In the case of the transfer learning model, it was trained over 5 epochs on the Lichess, set, and then over 5 epochs of the TCEC data. As is evident from the results, this particular case of transfer learning was detrimental to the performance of the baseline evaluation, so two other examples were tested as well.

The rationale after the first example (table 2.2) was that the model was being overfitted on the engine data, and therefore losing the fundamental chess knowledge. Therefore the engine data was trained on over one epoch, to reduce the total effect on the model. The difference between the number of draws in the two sets was motivation for the second example (table 2.3).

For the transfer learning implementation, the draws were eliminated from the engine data, and fitted over five epochs on top of the Lichess set. The results are shown in Table 2.5. These result paints the engine chess in a poisonous light. This may be because the nature of engine

Table 2.2: Baseline performance when trained of TCEC games

Seed	Training Accuracy	Validation Accuracy	Normalized Validation Accuracy	Use Case Accuracy	Normalized UCA
A	62.48%	17.80%	35.42%	55.83%	40.97%
B	62.55%	19.78%	36.24%	55.99%	40.46%
C	62.41%	17.84%	36.54%	55.56%	40.54%
	Win Accuracy	Draw Accuracy	Lose Accuracy	Leading Accuracy	Losing Accuracy
A	15.49%	78.21%	12.55%	54.67%	63.32%
B	19.29%	75.97%	13.45%	63.59%	54.95%
C	15.61%	82.25%	11.77%	64.03%	53.87%
	Win Count	Draw Count	Lose Count	Score	Elo Rating
A	17	3	480	-463	1220
B	5	5	490	-485	1004
C	9	1	490	-481	1106

Table 2.3: Baseline performance when trained on Lichess games

Seed	Training Accuracy	Validation Accuracy	Normalized Validation Accuracy	Use Case Accuracy	Normalized UCA
A	59.53%	59.87%	42.8%	59.95%	42.98%
B	59.61%	59.25%	43.1%	59.33%	43.04%
C	59.61%	59.24%	42.88%	59.29%	42.85%
	Win Accuracy	Draw Accuracy	Lose Accuracy	Leading Accuracy	Losing Accuracy
A	61.68%	1.7%	65.03%	61.73%	65.08%
B	58.55%	3.29%	67.45%	58.66%	67.57%
C	53.2%	2.88%	72.56%	53.28%	72.62%
	Win Count	Draw Count	Lose Count	Score	Elo Rating
A	386	14	100	286	2034
B	331	16	153	178	1934
C	403	10	87	316	2066

chess tournaments causes games to play out along the same lines, and when a position from a different line is presented to the model, it is too far removed from the data which the model has been fitted on. This might be rectified by presenting a larger training set of engine chess to the model, but for the sake of this research, a spade was called a spade, and the engine data was rejected.

Finally, each of the baselines are compared to Stockfish in Table 2.6 (The combination of com-

Table 2.4: Baseline performance when using transfer learning

Seed	Training Accuracy[6]	Validation Accuracy	Normalized Validation Accuracy	Use Case Accuracy	Normalized UCA
A	61.48%	22.99%	37.35%	57.50%	41.43%
B	61.37%	21.39%	36.96%	57.29%	41.29%
C	61.32%	23.42%	37.73%	57.40%	41.33%
	Win Accuracy	Draw Accuracy	Lose Accuracy	Leading Accuracy	Losing Accuracy
A	23.82%	73.4%	16.04%	68.06%	53.79%
B	17.03%	74.7%	19.16%	60.57%	60.78%
C	21.08%	72.13%	19.98%	60.72%	60.61%
	Win Count	Draw Count	Lose Count	Score	Elo Rating
A	75	6	419	-344	1501
B	69	5	426	-357	1484
C	71	8	421	-350	1491

Table 2.5: Baseline performance when using additional transfer learning techniques

Technique	Training Accuracy	Validation Accuracy	Normalized Validation Accuracy	Use Case Accuracy	Normalized UCA
One Epoch	57.44%	22.65%	37.98%	58.28%	42.06%
No Draws	76.6%	57.52%	40.71%	57.56%	40.95%
	Win Accuracy	Draw Accuracy	Lose Accuracy	Leading Accuracy	Losing Accuracy
One Epoch	21.52%	75.36%	17.06%	66.41%	57.38%
No Draws	62.39%	0%	59.75%	62.39%	59.75%
	Win Count	Draw Count	Lose Count	Score	Elo Rating
One Epoch	212	12	276	-64	1754
No Draws	78	4	418	-340	1508

puter and human chess training games denoted as *Cyborg chess*). From this table it is extremely interesting to observe the negative correlation between draw accuracy and the Elo rating. It is also promising to observe that there is a correlation between being able to predict the outcome of a chess game based on a position, and real world chess performance.

Table 2.6: Comparison of different baseline approaches with Stockfish

Technique	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
Engine chess	18.48%	36.07%	16.8%	12.59%	78.81%	1131
Lichess	59.45%	42.93%	57.81%	68.34%	2.62%	2007
Cyborg chess	22.60%	37.84%	20.64%	18.34%	73.41%	1492
Stockfish	62.70%	44.31%	69.36%	61.11%	2.45%	±3400 [27]

At the end of this chapter, we have developed software that can consume an API[7] to a neural network that will enable it to play chess. We also have a baseline model (neural network) that is fairly capable at predicting the outcome of a chess match. When combining these two, we have a chess capable agent, and a playground to experiment on.

[7] Application Programming Interface

Chapter 3

Manual input encodings, embeddings and datasets

In this chapter, the effect of the format of the data presented to the model is investigated. Manual encodings, trained embeddings and smaller datasets are applied and the impact on classification performance and chess capability is ascertained.

3.1 Relevant literature

According to [29], convolution operations in a neural network are not designed to be used on sparse data. It is therefore worth it to investigate the effect of removing the sparsity before presenting it to the convolution kernels. Furthermore, [30] states that removing games that resulted in a draw from their training data yielded improvements in chess performance. In the Lichess dataset, draws make up about 5.9% of the data, so the effect will likely not be large. It is still worth investigating, since other datasets, mainly the computer chess datasets, will contain a larger number of draws.

Another interesting and related idea stems from word embeddings. Word embeddings are used in natural language processing to keep words which are similar to one-another in meaning,

close to one-another in representation. This idea can be applied to chess too [30], since the encoding used on the data can have an effect on neural network based classifiers [31]. Usually one would want to avoid the use of ordinal values to represent categorical data, due to the effect that larger orders have on the numerical operations inside the model. By moving slightly away from the zero knowledge trend where chess engines have to learn everything from with zero-prior knowledge, ordinal values can be applied to chess since some pieces are inherently more powerful than others, and it is widely accepted that some pieces are simply worth more [32].

The final area of interest lies in the training set size. There is evidence that training set size has an effect of classification of images [33], and with the absolutely vast amount of training data available for chess, it is well worth investigating.

3.2 Addressing sparsity

This section will document the effect of sparse data at the input of the network on classifier accuracy and chess performance.

3.2.1 Experimental Setup

To investigate the absence of sparsity, a single dense layer, with 768 nodes, was introduced to the the input side of the network. This dense layer was reshaped[1] back into the original 12x8x8 configuration, to preserve some sense of dimensionality. The addition of the dense layer introduced 590,592[2] additional parameters to the network. This dense layer is effectively an autoencoder, since it encodes the chess position in an (almost completely) [3] unsupervised manner. To control for these extra parameters, the same amount of parameters was added to a different network, but after the convolution layers. See tables 3.1 and 3.2.

Hypothetically, the dense data at the convolution kernels should improve the accuracy of the network, which in turn should have a positive effect on the chess capability of the engine in which this model is deployed. While the addition of parameters should also improve the classification accuracy, especially in a network as small as the baseline, the gains achieved in the

[1] Reshaping refers to the operation that rearranges, for example, a 1x9 matrix into a 3x3 matrix.

[2] 768 * 768 weights connecting the inputs to the dense layer, plus the 768 biases in the new nodes is 590,592

[3] Almost completely since the supervised optimization of the entire model will have an effect on the first layer, but the objective function is wholly unrelated to chess encodings

model where convolution is not applied to sparse data should be more.

The architectures used for each of the models evaluated are shown in tables 3.1 and 3.2. Both the networks were trained on 430,000 human chess games, composed of about 28.6 million different positions. They were trained over five epochs of the data, using mini-batches of length 256 and an adaptive learning rate optimizer (ADADelta). Both were evaluated on 58,500 human chess games, with an average of 3.6 million different positions.

Table 3.1: Architecture for dense convolutional network

Layer (type)	Output Shape	# of Params	Activation
Dense	(768)	590592	ReLU
Reshape	(12,8,8)	None	—
Conv2D	(12, 8, 32)	2336	ReLU
Conv2D	(12, 8, 32)	9248	ReLU
Conv2D	(12, 8, 32)	9248	ReLU
Max Pooling	(6, 4, 32)	0	—
Dropout	(6, 4, 32)	0	—-
Flatten	(768)	0	—
Dense	(64)	49216	ReLU
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		660835 0 660835	

Table 3.2: Architecture for sparse convolutional network with added parameters

Layer (type)	Output Shape	# of Params	Activation
Conv2D	(12, 8, 32)	2336	ReLU
Conv2D	(12, 8, 32)	9248	ReLU
Conv2D	(12, 8, 32)	9248	ReLU
Max Pooling	(6, 4, 32)	0	—
Dropout	(6, 4, 32)	0	—-
Flatten	(768)	0	—
Dense	(768)	590592	ReLU
Dense	(64)	49216	ReLU
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		660,835 0 660,835	

3.2.2 Results

The networks in question were trained over three random seeds. They were evaluated in terms of classification accuracy per class, and chess playing capability.

The average results (the average of the results for a specific architecture, across the three seeds), were computed and is displayed in table 3.3. Results of specific seeds are shown in appendix A.1.

Table 3.3: Comparison of sparse and dense models with the baseline

Technique	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
Dense data	59.98%	44.16%	62.43%	64.57%	1.47%	2114
Sparse data	59.14%	42.41%	69.00%	56.73%	1.51%	1923
Baseline	59.45%	42.93%	57.81%	68.34%	2.62%	2007

3.2.3 Analysis of results

To assess whether or not the results listed in the previous section are statistically significant, a two-sample t-test was performed on the Elo rating achieved by each agent. Assuming that the null hypothesis was that the baseline and both the sparsely fed and densely fed networks are equal in chess playing performance, the t-test (with threshold $\alpha = 0.1$) yielded the following:

Table 3.4: Statistical comparison between the sparse, dense and baseline networks

Null Hypothesis	t-statistic	p-value	Verdict
The sparse network is different from the baseline	1.416	0.292	Not rejected
The sparse network is different from the dense network	-3.442	0.075	Rejected
The dense network is different from the baseline	-2.038	0.178	Not rejected

which suggests that the dense convolutional network is very likely to be better at chess than the baseline, and the sparse convolutional network is likely worse. The dense network is almost certainly better than the sparse counterpart.

There is no statically significant evidence that the introduction of the dense layer improved performance. It is still likely though, that as hypothesised, the addition of a dense layer before the convolution kernels did have a positive effect on both classification and chess performance. This must be almost wholly attributed to the absence of sparsity, since unlike hypothesised, the addition of parameters elsewhere in the network, had an adverse effect on both classification

and chess capability.

The results of this particular experiment seems to provide evidence that performing convolution on dense data, even if the density was achieved somewhat artificially, does improve the generalizing ability of the network.

3.3 Addressing dimensionality

This section will document the effect of manual piece encodings and representational embeddings of chess positions on chess performance and classification accuracy. Throughout this section, a particular distinction between an encoding and an embedding should be noted. Encodings refer to manually encoded descriptions of chess postitions, while (word)-embeddings refer to trained encodings of chess positions.

3.3.1 Experimental Setup

The experiments detailed in this section depends greatly on the writer's creative ability to represent chess positions in meaningful ways. Keep in mind that chess is played with 32 pieces. These pieces are divided into two sides. Each side has six different pieces, with two of each piece, with the exception of the King and Queen, which are singletons, and the pawns, of which there are eight.

The two encodings relevant to this section was derived from the fact that there needs to be a distinction between two opposing sides.

For the first encoding, these two sides were kept apart, each inhabiting it's own 8x8 matrix. Consider the matrix W (equation 3.1), showing the location of the white pieces. Let the ranks (rows) and files (columns) be represented by *i,j* respectively.

Let each piece be assigned an ordinal value, based on how much it is worth, pawns one, through the king, at six. If a zero represents an empty square, then the opening position for

white is described by:

$$\mathbf{W} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 2 & 3 & 5 & 6 & 3 & 2 & 4 \end{bmatrix} \tag{3.1}$$

.

Similarly, the starting position for the black pieces, B, will look like:

$$\mathbf{B} = \begin{bmatrix} 4 & 2 & 3 & 5 & 6 & 3 & 2 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.2}$$

.

To describe the position of the pieces completely, these two matrices will be stacked, to form an 2x8x8 tensor. The position shown in figure 3.1 is represented in figure 3.2.

Keen eyed observers might notice that there are quite a lot of sparsity in the previous representation. One way to reduce the empty space in the tensor, would be to represent pieces of opposing colour with opposing signs. In keeping with the higher values for more powerful pieces scheme, this fits quite well, since the most powerful black piece poses the largest threat

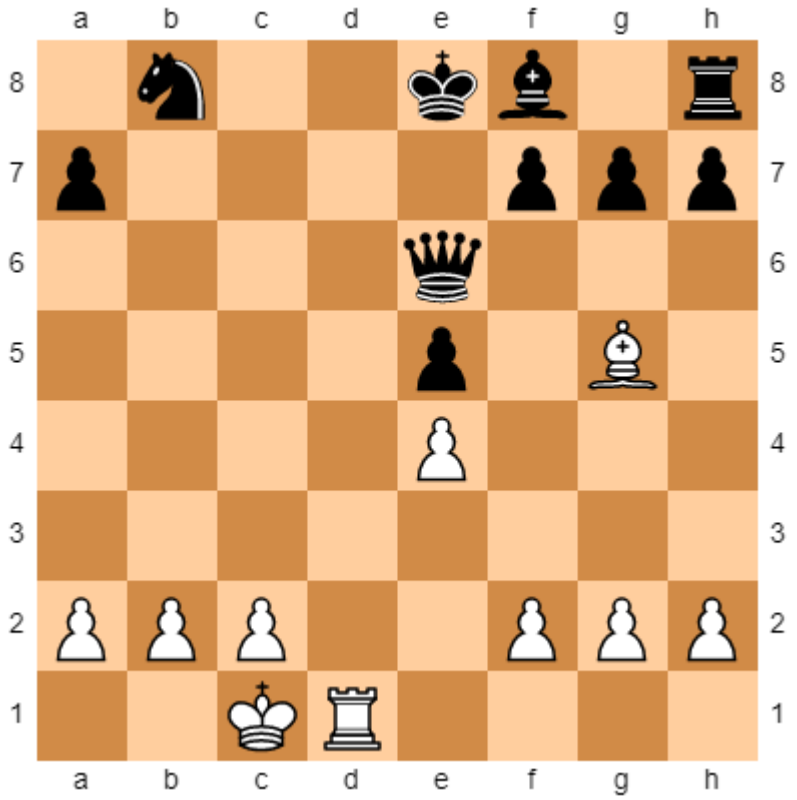

Figure 3.1: Chess match between Paul Morphy vs Duke Karl

to white and vica versa. The position shown in figure 3.1 will then be described by P:

$$\mathbf{P} = \begin{bmatrix} 0 & -2 & 0 & 0 & -6 & -3 & 0 & -4 \\ -1 & 0 & 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & -5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 6 & 4 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.3}$$

These two representations on the architectures described in tables 3.5 and 3.6. The only difference between these two evaluation functions is the way the input data is presented, but from the tables it can be seen that this has a profound effect on the number of parameters that have be trained. As a reminder, the baseline model consists of 70,243 parameters.

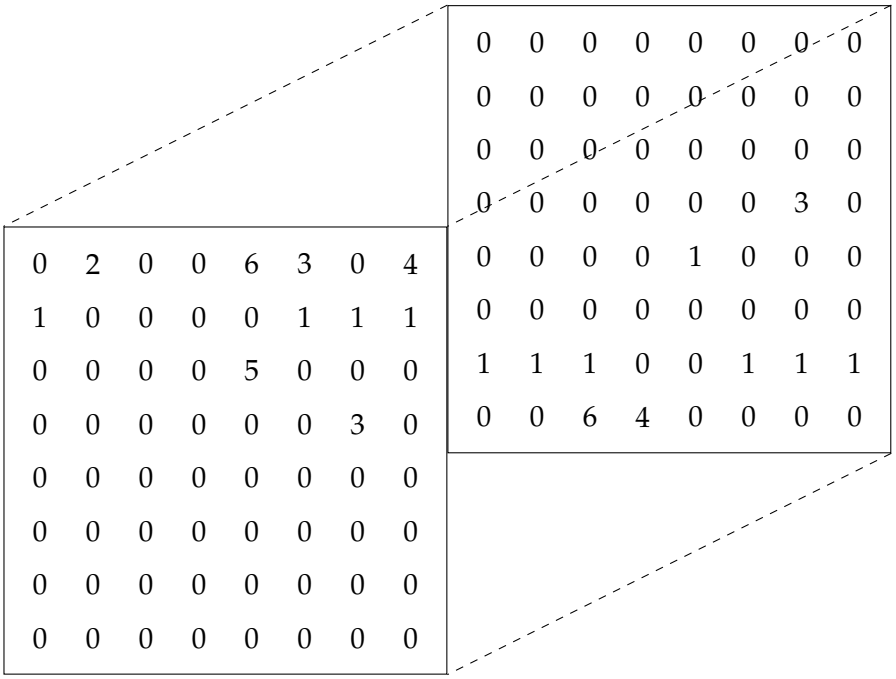

Figure 3.2: 8x8x2 representation of chess pieces

Table 3.5: Architecture for 8x8x2 encoding

Layer (type)	Output Shape	# of Params	Activation
Conv2D	(2, 8, 32)	2336	ReLU
Conv2D	(2, 8, 32)	9248	ReLU
Conv2D	(2, 8, 32)	9248	ReLU
Dropout	(6, 4, 32)	0	—-
Flatten	(512)	0	—
Dense	(64)	32832	ReLU
MaxPooling	(1,4,8,32)	0	—
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		53859 0 53859	

The development of word-embeddings will now be considered. In an effort to keep the amount of parameters equal, the embeddings mapped the original data encoding to the same dimensions as the manual encodings. At this point, it is worth noting that an embedding, as discussed in this research, serves only the purpose of reducing the dimensions of the input to the classifier. A side-effect of the way that these embeddings go about decimating dimensions, is that the sparse nature of the input data is completely removed. In essence, the ability of an

Table 3.6: Architecture for 8x8x1 encoding

Layer (type)	Output Shape	# of Params	Activation
Conv2D	(1, 8, 32)	2336	ReLU
Conv2D	(1, 8, 32)	9248	ReLU
Conv2D	(1, 8, 32)	9248	ReLU
Dropout	(6, 4, 32)	0	—-
Flatten	(768)	0	—
Dense	(64)	16448	ReLU
Dropout	(64)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		37475 0 37475	

unsupervised[4] dense layer to encode a chess position is compared to meaningful handcrafted encodings. The architecture for these models are displayed in tables 3.7 and 3.8.

Table 3.7: Architecture for 8x8x1 embedding

Layer (type)	Output Shape	# of Params	Activation
Dense	(768)	590592	ReLU
Reshape	(2,8,8)	None	—
Conv2D	(12, 8, 32)	2336	ReLU
Conv2D	(12, 8, 32)	9248	ReLU
Conv2D	(12, 8, 32)	9248	ReLU
Max Pooling	(6, 4, 32)	0	—
Dropout	(6, 4, 32)	0	—-
Flatten	(768)	0	—
Dense	(64)	49216	ReLU
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		660835 0 660835	

The hypothesis with regards to the manual encodings was that they should not fare better than the baseline, since there is sufficient information for the baseline to approximate piece values. With regards to the embeddings, the elimination of the sparsity in itself should mean that the overall classifier performance will be higher than the baseline. The effect of a reduction in dimensionality can be assesed against the dense convolutional network described in section3.2.

[4]These layers are unsupervised since, at no point are there any clues about how to encode the chess position in lower dimensions

Table 3.8: Architecture for 8x8x2 embedding

Layer (type)	Output Shape	# of Params	Activation
Dense	(768)	590592	ReLU
Reshape	(1,8,8)	None	—
Conv2D	(12, 8, 32)	2336	ReLU
Conv2D	(12, 8, 32)	9248	ReLU
Conv2D	(12, 8, 32)	9248	ReLU
Max Pooling	(6, 4, 32)	0	—
Dropout	(6, 4, 32)	0	—-
Flatten	(768)	0	—
Dense	(64)	49216	ReLU
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		660835 0 660835	

The embeddings should have a positive effect on chess performance because of its inate ability to represent a chess position in a way that will reduce overall model error, and it was therefore expected to outperform the dense convolutional model.

3.3.2 Results

As per usual, these networks were trained across 3 random seeds, over 5 epochs. The training set consisted of 88% of the data, and the model was validated on the remaining 12%. The results, averaged over all the seeds, are displayed in table 3.9 In this table the "Dense data convolutional" network refers to a network from the previous section (3.3.1). For a more detailed breakdown of how specific seeds performed, refer to appendix A.1.

Table 3.9: Comparison of embeddings, dense convolutional and the baseline

Technique	Validation Acc.	Normalized VA	Win Acc.	Loss Acc.	Draw Acc.	Elo Rating
8x8x2 Embedding (Table 3.7)	60.41	43.86	60.25	67.62	3.70	2227
8x8x1 Embedding (Table 3.8)	60.32	43.80	60.91	66.63	3.87	2237
Dense data convolutional	59.98	44.16	62.43	64.57	1.47	2114
8x8x2 Manual (Fig. 3.1)	58.78	41.97	62.43	56.77	1.39	1988
8x8x1 Manual (Eq. 3.3)	58.71	42.28	69.00	60.54	2.33	1981
Baseline	59.45	42.93	57.81	68.34	2.62	2007

3.3.3 Analysis of results

For this analysis, hypothesis tests were conducted to assert the following:

- whether the manual encodings perform better than the baseline
- if there is evidence to suggest that one manual encoding is better than the other
- whether or not the embeddings perform better than the dense convolutional model
- and whether a specific embedding is superior

These results are listed in table 3.10. The null hypothesis will be evaluated against a threshold of $\alpha = 0.1$, which yields a confidence interval of 90%. The t-tests were conducted on the Elo rating of the chess agent.

The statisctical analysis yielded some interesting results. As expected, the evidence suggests that there is no discernible difference between the manual encodings and the baseline. This carries the implication that the baseline is apt at approximating piece power or value. Regarding the difference between the two encodings, the t-test suggests that there is a 90% chance that there are no differences between the encodings, but since neither of them really differ from the baseline this is a moot point.

Table 3.10: Statistical comparison between manual encodings, embeddings and the baseline

Null Hypothesis	t-statistic	p-value	H_0
The 8x8x2 encoding is not different from the baseline	0.3772	0.74	Not rejected
The 8x8x1 encoding is not different from the baseline	0.4901	0.67	Not rejected
The 8x8x1 encoding is not different from the 8x8x2	0.145	0.90	Not rejected
The 8x8x2 embedding does not differ from the dense-conv model	2.07	0.17	Not Rejected
The 8x8x1 embedding does not differ from the dense-conv model	3.31	0.08	Rejected
The 8x81 embedding does not differ from the 8x8x2	0.21	0.85	Not rejected

A really interesting results can be observed from the embeddings. Although there is not entirely enough evidence to suggest that the larger of the two embeddings (the 8x8x2 one) is better than the dense convolutional model, the 8x8x2 embedding is almost certainly better. There is also little evidence to suggest that there is a difference between the two embeddings, so it is safe to conclude that using some sort of dense embedding is beneficial to chess performance.

When considering classifier performance, the validation accuracy of the manual encoded networks were lower than that of the baseline, which bodes well for the one-hot encoding arguments. Once again, the embeddings have impressed, showing an increase in validation accuracy over not only the baseline, but the dense convolutional model as well. This would suggest that manual encodings worsened these networks' generalizing ability and embeddings enhanced it.

3.4 Addressing different data sets

This section serves to document the impact of different datasets on classifier performance. The main area that will be investigated is the size of sets, but a small detour on class distribution is entertained as well.

3.4.1 Experimental setup

At the time of writing, Lichess had a database consisting of 1.6 billion different standard[5] chess games. This database has, since its inception, seen exponential growth. As of October of 2020, about 70 million games are added per month. This begs the question whether or not a larger dataset will yield a better chess classifier. Due to real world constraints, like time and money, the majority of the research documented here was done on only 500,000 games, less than 0.1% of the total games available. Due to the same constraints, the effect of more chess games was emulated by observing the effect less games had, and then extrapolating.

This was done by effectively halving the amount of data available to the classifier a number of times, and documenting the effect. To investigate the effect of dataset size, the classifier was trained on the number of games listed in table 3.11. On average, a game consisted of 67 plies (positions), which yielded the number of samples that every network was trained on.

[5]Lichess keeps record of different chess variants as well

Table 3.11: Training set sizes

Designator	Games in training set	Positions in training set	Games in validation set	Positions in validation set
Baseline	429,000	28,646,560	58,500	3,906,000
Less chess	214,500	14,323,280	58,500	3,906,000
Even less chess	117,000	7,812,698	58,500	3,906,000
Almost no chess	58,500	3,906,337	58,500	3,906,000
No draw	403,543	26,946,720	58,500	3,906,000

Table 3.12: Dataset distribution with respect to the white pieces

Win	Draw	Loss
46.69%	5.93%	47.37%

As mention in Section 3.1, there might be some merit in omitting games that resulted in a draw. The effect of this omission might be larger on sets where the number of draws are higher, (the CCL and TCEC datasets, for example), but it will nevertheless be investigated here. The class distribution of the lichess dataset is depicted in table 3.12. The ”no-draw” classifier was therefore trained on approximately 404,000 games.

The effect of omitting resultless games was expected to have negligible impact on chess performance, and a small effect on classifier performance. This small effect stems from the fact that the classifier will be evaluated on data from a class, where the class has never been presented to it. The effect of more data was expected to yield diminishing returns [20] [33], across chess performance and classifier accuracy. This should be attributed to the reduction in novel samples as the dataset gets ever larger.

3.4.2 Results

The performance of each classifier is compared to the baseline in table 3.13. These results are obtained by averaging over all three seeds for each specific setup. The details of specific seeds can be viewed in appendix A.1.

3.4.3 Analysis of results

The first result to be analysed was the difference between the baseline and the model trained without draws. A two-sample t-test was used, with the null hypothesis being that there is no difference between the classifiers in question. The metric used for the t-tests were the Elo rating

Table 3.13: Comparison of different data sets to the baseline

Designator	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
Almost no chess	57.44%	42.58%	65.86%	67.62%	6.51%	1680
Even less chess	59.22%	43.03%	62.59%	66.63%	3.57%	1911
Less chess	59.34%	42.81%	57.61%	64.57%	2.6%	1914
No draw	59.38%	42.14%	63.96%	62.47%	0%	2004
Baseline	59.45%	42.93%	57.81%	68.34%	2.62%	2007

obtained whilst using each model. To establish whether the difference between the reduced dataset classifiers and the baseline was statistically significant, t-tests were used as well. The tests were conducted with a threshold value of $\alpha = 0.1$, for a confidence interval of 90%, and the verdicts of these tests are listed in table 3.14.

Table 3.14: Statistical comparison between different datasets and the baseline

Null Hypothesis	t-statistic	p-value	H_0
[No draw] does not differ from the baseline	0.0456	0.9678	Not rejected
[Almost no chess] does not differ from the baseline	6.533	0.0226	Rejected
[Even less chess] does not from the baseline	2.085	0.17243	Not rejected
[Less chess] does not differ from the baseline	1.624	0.2457	Not Rejected

Although the hypothesis tests suggest that only one of the results are statistically significant, there is more to it than the p-values would lead one to believe. From this analysis it is safe to assume that the omission of the drawn games from the training data made no difference. Further, while not completely significant, the reduction in dataset size nudged the results closer and closer to a "significant" value, which implies that there is a relation between chess performance, and the amount of data available to train on. The gains in chess performance seem logarithmic, initially at first, which means that given enough time and money, anything is possible.

Figure 3.3 shows this relation visually, while figure 3.4 shows the relation between validation accuracies and dataset sizes.

Although the gains in accuracy are small, they are still present, confirming the hypothesis.

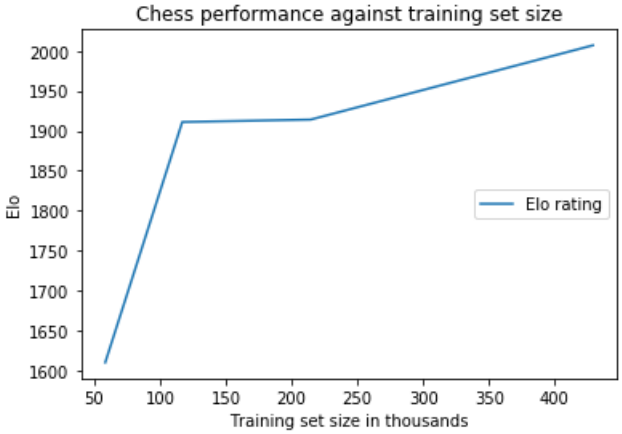

Figure 3.3: Chess performance against training set size

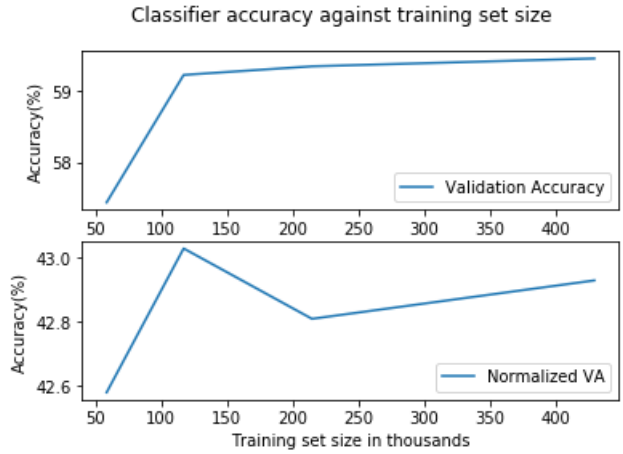

Figure 3.4: Classifier performance against training set size

3.5 Concluding remarks

This chapter investigated the effect of manual input encodings, embeddings and datasets on a classifier used as an evaluation function in a chess engine. A few interesting phenomena could be observed. Each area yielded a positive result, in one form or another, but the most significant gains were achieved by removing the sparsity from the input data. In this research, this was done by an autoencoder in the form of a dense layer after the input layer, but other means may yield a similar result. This confirms the findings by [29].

Using manual encodings to represent chess pieces did not have any effect on chess capability, but it did have an adverse effect on overall classifier accuracy, in such it reduced the classifier's

ability to generalize, which is a nod in the direction of one-hot encodings.

Instead of using manual encodings, a designer of a chess engine should rather consider an embedding to reduce dimensionality. Not only did this yield the best chess and classifier performance of everything evaluated in this chapter, the reduction in dimensionality also reduced the computational complexity of the evaluation[6].

Finally, the effect of larger datasets provided a promising glimpse into performance gains. If the performance gains are indeed logarithmic, a chess engine could achieve state of the art performance when using all the games from the Lichess database alone. This is, however, extremely optimistic, since [3] shown that after a certain number of games, actual chess performance gains tapers off drastically.

All things considered, it is worth it to pay attention to all things data when training any sort of classifier, with a chess evaluation function being no exception.

[6]This reduction is in comparison to using only an autoencoder to remove sparsity

Chapter 4

Network anatomy and topology

In this chapter, the effect of the network anatomy (the types of ”neurons” used in the neural network) and topology (the depth and width of the network) on the chess evaluation function is investigated. A brief overview of relevant literature is provided before convolution kernels are compared to dense layers. Various width and depth configurations are evaluated for both convolutional and dense networks. This is all in a quest to evaluate chess capabilities and classifier performance.

4.1 Relevant literature

One of the objectives of this chapter is to determine how wide or deep the neural network classifying chess positions have to be. The ImageNet Large Scale Visual Recognition Challenge is an annual competition for algorithms in object detection and image recognition. The competition has been held since 2010, and since the first Convolutional Neural Network — eight layers deep – won, the best classifiers has gained a substantial amount of layers. The winners since 2015 have had upwards of 150 layers [34]. Clearly deeper convolutional networks are better at object detection, begging the question if the same logic can be applied to chess positions.

Another pressing matter when designing a neural network classifier, is the selection of hy-

perparamters. Unfortunately, according to [35], setting hyperparameters in convolutional networks (kernels size, number of features, etc.) is still an open problem. They achieved promising results using genetic algorithms to tune hyperparameters. This seems to be less of an obstacle in dense networks, as [36] claims that network depth contribute much more to classifier performance than width. Work by Lu [37] also investigate the effect of width in dense networks, claiming that one needs a network width of $n+4$ to approximate any function, with n being the number of input dimensions.

Work on object detection in video have shown progress using three dimensional convolutional networks. Results by [38] and [39] are promising on real-time object detection, while [40] did work on hand gesture recognition. This is a viable approach for chess as well, due to the three dimensional nature of the chess representation developed in chapter 2.

4.2 Investigating depth in convolutional networks

In this section, the effect of depth, or the amount of subsequent layers, in a convolutional neural network will be investigated when said network is used as an evaluation function for a chess engine.

4.2.1 Experimental setup

In practice, a neural network rarely consists of exclusively convolution kernels, since the innate reduction in dimensionality that a classifier usually needs to do, isn't accomplished well with the convolution operation. The baseline model, discussed in Chapter 2, is no exception. It consists of three convolution layers, followed by two dense layers, which mainly serves the purpose of reducing the data from dimension $n=768$, to $n=3$, for every output class that the classifier can classify.

While there are many ways to describe the depth of a neural network, the depth referred to in this section, will refer to the number of convolution layers in the classifier. To illustrate, while the baseline consists of six hidden layers, it had a depth $d=3$, since it only had three convolution layers. Note that this definition serves no purpose other than to describe the amount of convolutional layers in the classifier.

Networks with five (the baseline and four others) different depths will be investigated. Table 4.1 shows a summary of these networks, their depths and the number of parameters each consists of. Table 4.2 shows a brief summary of the architecture used for the 1 layer variant, with the layer that will be repeated in subsequent networks highlighted. For an overview of what unclear terms might describe, consult the Deep Learning Book by Goodfellow et al. [23].

Table 4.1: Architectures, depth and parameters of convolutional networks

Architecture	Depth	Parameters
Convolutional	1	51,747
Convolutional	2	60,995
Baseline	3	70,243
Convolutional	4	79,941
Convolutional	5	88,793

Table 4.2: Architecture of 1-layer convolutional network

Layer (type)	Output Shape	# of Params	Activation
Conv2D	(12, 8, 32)	2336	ReLU
Max Pooling	(6, 4, 32)	0	—
Dropout	(6, 4, 32)	0	—-
Flatten	(768)	0	—
Dense	(64)	49216	ReLU
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		51747 0 51747	

Convolutional networks generally fare well when the data they are used on has some form of spatial relationship. As demonstrated in the previous chapters, the convolution kernels in the baseline fare well when presented with a chess position in the form that has been described. The literature predicts that deeper convolutional networks fare better as classifiers, and it is therefore expected that deeper networks will fare better in chess as well [41].

4.2.2 Results

The networks were trained on three random seeds and over five epochs, using mini-batches of length 256 and an adaptive learning rate optimizer. The average result of each architecture is listed in table 4.3. For a more detailed breakdown of the results of each seed for all the networks, refer to Appendix A.2.

Table 4.3: Comparison of convolutional networks of different depths

Architecture	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
1 layer Convolutional	59.20%	42.47%	59.08%	66.53%	1.82%	1910
2 layer Convolutional	59.36%	42.87%	64.94%	60.93%	2.76%	1910
Baseline	59.45%	42.93%	57.81%	68.34%	2.62%	2007
4 layer Convolutional	59.38%	43.13%	59.73%	66.44%	3.23%	2030
5 layer Convolutional	59.47%	43.67%	58.43%	67.32%	5.24%	2097

4.2.3 Analysis of results

In an attempt to gauge how much of the observed results are due to chance or noise, two-sample t-tests were conducted on the Elo rating achieved. In these hypothesis tests, every network was compared to the baseline, to assess their performance relative to other classifiers in this dissertation. The effect of depth was analysed by comparing the single layer network to the deepest one. Table 4.4, shows these results. The verdicts of these tests are based on a confidence interval of 90% which causes the threshold α to be 0.1.

Table 4.4: Statistical comparison between convolutional networks of different depths

Null Hypothesis	t-statistic	p-value	H_{null}
1-layer convolution is similar to the baseline	-2.0982	0.1708	Not rejected
2-layer convolution is similar to the baseline	-1.53	0.2653	Not rejected
4-layer convolution is similar to the baseline	0.5584	0.6327	Not rejected
5-layer convolution is similar to the baseline	1.7742	0.2180	Not rejected
1-layer convolution is similar to the 5-layer	4.7575	0.0415	Rejected

From these results, it is interesting to observe that none of the networks differs enough from the baseline to definitively say that one is better. It is interesting to note that the significance of the difference increases as the amount of layers differ from the baseline.

The t-test result also leads to the fact that, there is almost no doubt that the 5 layer network is better than the 1 layer network. This carries the implication that an increase in depth does have a positive effect on chess capabilities. The following figures demonstrates this visually.

Figures 4.1 and 4.2 visually shows the effect that depth imposes on the classifier in this setup. An upwards trend can be observed in the relation between the depth of the network and its chess capabilities, and a fairly linear trend at that. It would be naive to believe that this linear

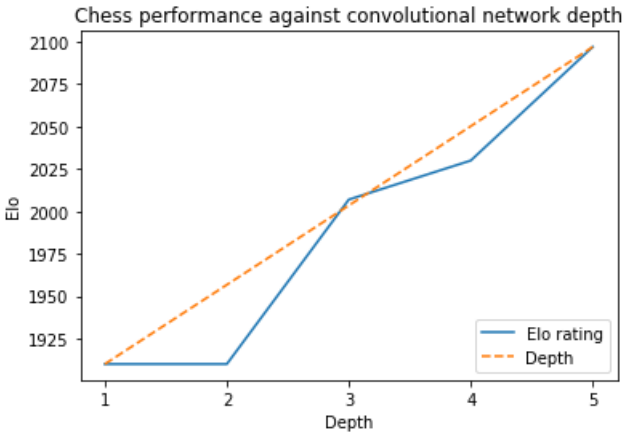

Figure 4.1: Chess performance against convolutional network depth

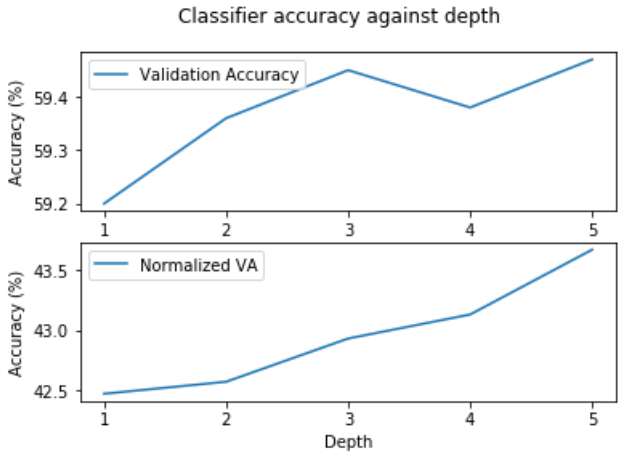

Figure 4.2: Classifier accuracy against convolutional network depth

trend would continue indefinitely, but this results show that there is merit in increasing depth in pursuit of a better chess evaluation function.

4.3 Investigating width in convolutional networks

In this section, the effect of width in convolutional neural networks will be assessed in terms of classifier accuracy, and chess performance. In this section, width will refer to the amount of convolution kernels used in each layer.

4.3.1 Experimental setup

For this experiment, the baseline model was modified in only one aspect. The number of convolution kernels in all three layers was modified so that each layer has the amount that is specified.

Like in the previous section, five different architectures were investigated. The architecture of the smallest network, width $w = 8$, is explained in table 4.5. The relevant width is shown in the output shape column, given by the third entry (printed in bold here and in the table). The network with width 16 had output shapes of (12, 8, **8**) in the first three layers.

Table 4.5: Architecture of 1-layer convolutional network

Layer (type)	Output Shape	# of Params	Activation
Conv2D	(12, 8, **8**)	584	ReLU
Conv2D	(12, 8, **8**)	584	ReLU
Conv2D	(12, 8, **8**)	584	ReLU
Max Pooling	(6, 4, 8)	0	—
Dropout	(6, 4, 8)	0	—-
Flatten	(192)	0	—
Dense	(64)	12352	ReLU
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		14299 0 14299	

Table 4.6 shows a summary of the networks trained, their width, and the number of parameters they consist of.

Table 4.6: Architectures, width and parameters of convolutional networks

Architecture	Width	Parameters
Convolutional	8	14,299
Convolutional	16	30,643
Baseline	32	70,243
Convolutional	48	119,059
Convolutional	64	177,091

Unfortunately, to the extent of the writer's knowledge, there exists no "one-way fits all" approach to specify the number of kernels in convolutional networks. The best way to tune this hyper-parameter seems to be trial and error, which means that there isn't a rigorous way to

form an hypothesis about how width will affect performance.

As the width of the network increases, the number of trainable parameters increases as well, which should allow the classifier to pick up finer nuances in the data presented to it. The gains that are obtainable with this approach is capped by the complexity of the data, and therefore performance is expected to reach a peak, and then plateau with the addition of more kernels in each layer.

4.3.2 Results

The networks were trained on three random seeds and over five epochs, using mini-batches of length 256 and an adaptive learning rate optimizer. The average result of each architecture is listed in table 4.7. For a more detailed breakdown of the results of each seed for all the networks, refer to Appendix A.2.

Table 4.7: Comparison of convolutional networks of different widths

Architecture	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
8 wide Convolutional	59.05%	42.49%	62.89%	62.44%	2.16%	1912
16 wide Convolutional	59.32%	42.85%	62.29%	63.68%	2.59%	1935
Baseline	59.45%	42.93%	57.81%	68.34%	2.62%	2007
48 wide Convolutional	59.57%	43.39%	66.42%	59.78%	3.98%	1915
64 wide Convolutional	59.13%	43.02%	67.85%	58.03%	3.17%	1926

4.3.3 Analysis of results

The results listed in table 4.7 shows that every model tested fared worse than the baseline. To establish whether this is due to variance, two-sample t-tests were conducted on the Elo rating obtained by the various classifiers. The results of these tests are shown in table 4.8. The verdicts of these tests are delivered based on a threshold value of $\alpha = 0.1$, for a confidence interval of 90%.

The hypothesis tests shows that there is some, although not enough to be statistically significant, evidence to suggest that every single one of the networks performed worse than the baseline. The comparison between the 8 kernel and 48 kernel network shows that there is likely no effect to be observed when changing the width of a convolutional network.

Table 4.8: Statistical comparison between convolutional networks of different widths

Null Hypothesis	t-statistic	p-value	H_{null}
8-wide convolution is similar to the baseline	-1.9532	0.190	Not rejected
16-wide convolution is similar to the baseline	-1.5423	0.2629	Not rejected
48-wide convolution is similar to the baseline	-2.176	0.1615	Not rejected
64-wide convolution is similar to the baseline	1.937	0.1923	Not rejected
8-wide convolution is similar to 48-wide convolution	-0.4707	0.6842	Not rejected

The chess performance and classification accuracy is compared visually in the following figures.

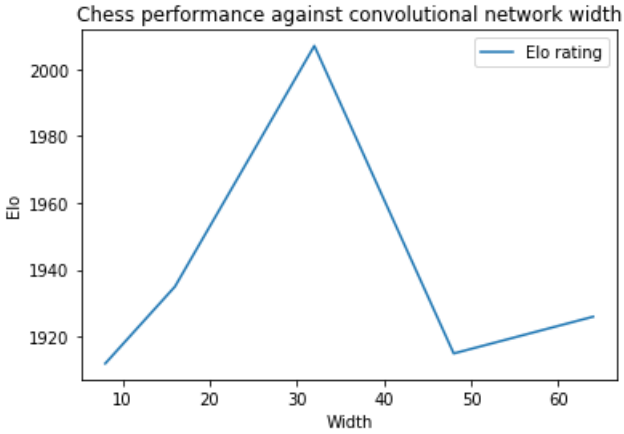

Figure 4.3: Chess performance against convolutional network width

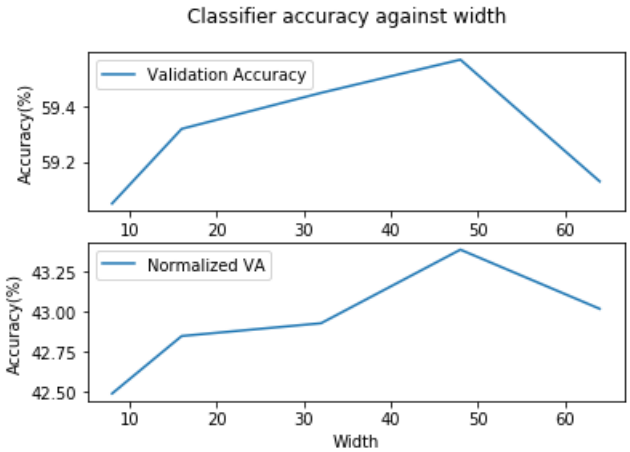

Figure 4.4: Classifier accuracy against convolutional network width

Figures 4.3 and 4.4 illustrates the absence of gain with the increase of width. The hypothesis that an increase in width will have a positive effect up to a point, can be accepted. The

interesting phenomena here is that additional kernels, past the ”sweet spot” seem to have a detrimental effect on both chess performance, and classifier accuracy. Considering the results of this section, it would seem that the optimal number of kernels to use in this classifier lies between 32 and 48. This number might change based on the representation of chess positions as well as the output of the classifier.

In summary, network width in convolutional networks is a hyper parameter that needs to be tuned differently for different applications and there is no recommendation that can be made as a rule of thumb.

4.4 Investigating alternatives to convolution

This section will investigate the capability of alternatives to the convolution operation in the evaluation function. At first, the convolution layers will be replaced with ordinary dense layers after which the viability of 3D-Convolution will be entertained.

To understand why 3D-convolution might be beneficial, it is necessary to first understand how 2D-convolution is done in data that has more than two dimensions, such as a three-channel image.

Consider first a single channel, gray scale image, **W** of dimension 4x4, and a 3x3 kernel **C** :

$$W = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix} \tag{4.1}$$

$$C = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \tag{4.2}$$

.

Element (1,1) of the convolution of **W** with **C**, can be computed by taking the dot product of **C** with the north-western 3x3 square of **W**.

This is equal to:

$$W_{11}^{*} = x_{11}u_{11} + x_{12}u_{12} + x_{13}u_{13} + x_{21}u_{21} + x_{22}u_{22} + x_{23}u_{23} + x_{31}u_{31} + x_{32}u_{32} + x_{33}u_{33} \quad (4.3)$$

Now, if **V** was an RGB image of dimension 3x4x4 and the entries into each channel of **V** was prepended with the channel name, the green channel of **V** is depicted as:

$$V_g = \begin{bmatrix} x_{g11} & x_{g12} & x_{g13} & x_{g14} \\ x_{g21} & x_{g22} & x_{g23} & x_{g24} \\ x_{g31} & x_{g32} & x_{g33} & x_{g34} \\ x_{g41} & x_{g42} & x_{g43} & x_{g44} \end{bmatrix} \quad (4.4)$$

Convolving **C** with the entirety of **V**, is done by convolving each channel separately, and then summing the results. For example, the (1,1) entry of the convolution of **C** and **V** is equal to:

$$\begin{aligned} &\mathrm{V}_{11}^{*} = \\ &(x_{r11}u_{11} + x_{r12}u_{12} + x_{r13}u_{13} + x_{r21}u_{21} + x_{r22}u_{22} + x_{r23}u_{23} + x_{r31}u_{31} + x_{r32}u_{32} + x_{r33}u_{33}) + \\ &(x_{g11}u_{11} + x_{g12}u_{12} + x_{g13}u_{13} + x_{g21}u_{21} + x_{g22}u_{22} + x_{g23}u_{23} + x_{g31}u_{31} + x_{g32}u_{32} + x_{g33}u_{33}) + \\ &(x_{b11}u_{11} + x_{b12}u_{12} + x_{b13}u_{13} + x_{b21}u_{21} + x_{b22}u_{22} + x_{b23}u_{23} + x_{b31}u_{31} + x_{b32}u_{32} + x_{b33}u_{33}) \quad (4.5) \end{aligned}$$

Because of the way the convolution is computed in higher dimensional data, the parameters of each kernel can never be updated to consider patterns that span different channels, the result is always the superposition of a single feature in each channel individually.

This is where 3D-convolution is handy. Convolving a 3D kernel with 3D data computes a similar dot product to that shown in equation 4.5, except that the parameters of said kernel can be updated to give a higher activation if there is an interesting feature present in the combination of two or more layers. Consider what happens when **D** is the convolution kernel, comprised

of three versions of **C** stacked on top of each other. The convolution of **V** with **D** will then be the dot product of each element of the two tensors, sliding the kernel across three dimensions. The (1,1,1) entry of this convolution can be computed as follows:

$$
\begin{aligned}
&\text{V*D}_{111} = \\
&(x_{r11}u_{111} + x_{r12}u_{112} + x_{r13}u_{113} + x_{r21}u_{121} + x_{r22}u_{122} + x_{r23}u_{123} + x_{r31}u_{131} + x_{r32}u_{132} + x_{r33}u_{133}) + \\
&(x_{g11}u_{211} + x_{g12}u_{212} + x_{g13}u_{213} + x_{g21}u_{221} + x_{g22}u_{222} + x_{g23}u_{223} + x_{g31}u_{231} + x_{g32}u_{232} + x_{g33}u_{233}) + \\
&(x_{b11}u_{311} + x_{b12}u_{312} + x_{b13}u_{313} + x_{b21}u_{321} + x_{b22}u_{322} + x_{b23}u_{323} + x_{b31}u_{331} + x_{b32}u_{332} + x_{b33}u_{333})
\end{aligned}
\tag{4.6}
$$

Notice that the amount of operations involved to calculate a single entry is exactly the same, but that there is no collapse of dimensions in the 3 dimensional case. This means that in the next layer, there will be much more data still present in the hidden layers, causing 3D convolutional networks to be computationally (a lot) more expensive. A simple way to demonstrate the aforementioned fact is to consider that the output of the 2D convolution on 3D data is 2 dimensional, while the output of the 3D convolution on 3D data remains 3 dimensional.

4.4.1 Experimental setup

This section investigates two alternatives to conventional convolution. The first one is to just forgo convolution altogether, and the second is to consider convolutions of a higher dimension.

For this experiment, the 2D-convolution layers of the baseline was replaced by dense layers, each with a width similar to the dimensions of the input data. Then the convolution layers of the baseline was simply changed to consist of 3D kernels of the same number and dimension. The resulting architectures are displayed in tables 4.9 and 4.10.

Forming a hypothesis is once again not as simple as one would have hoped. The dense network is noticeably heavier than the baseline, consisting of about 30 times as many trainable parameters. The dense network does have the drawback of being agnostic to locality, which means that it might miss important features in data of this kind (pawn structure as a quick example).

Table 4.9: Architecture of 3-layer dense network

Layer (type)	Output Shape	# of Params	Activation
Dense	(768)	590592	ReLU
Dense	(768)	590592	ReLU
Dense	(768)	590592	ReLU
Dropout	(768)	0	—-
Dense	(64)	49216	ReLU
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		1,821,187 0 1,821,187	

Table 4.10: Architecture of 3D-convolutional network

Layer (type)	Output Shape	# of Params	Activation
Conv3D	(1,12, 8, 8)	6944	ReLU
Conv3D	(1,12, 8, 8)	27680	ReLU
Conv3D	(1,12, 8, 8)	27680	ReLU
Dropout	(1,12, 8, 8)	0	—-
Flatten	(3072)	0	—
Dense	(64)	196672	ReLU
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		259,171 0 259,171	

It was therefore hypothesised that the dense network would perform worse than the baseline in both chess performance and classifier accuracy.

Now, considering the 3D-convolutional networks have more parameters, and that they are able to train on interactions between different pieces in a way that their two-dimensional counter parts could not, they should perform better than the baseline in chess performance and classifier accuracy.

4.4.2 Results

The networks in question were trained on three seeds, and over five epochs. The various results of each seed was averaged and is shown in table 4.11. For more detailed results pertaining to

each seed, consult appendix A.2.

Table 4.11: Comparison of alternative network anatomies

Architecture	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
3-layer Dense	58.82%	43.39%	62.23%	62.02%	5.94%	1700
3D-Convolutional	59.83%	44.05%	60.41%	66.33%	3.22%	2020
Baseline	59.45%	42.93%	57.81%	68.34%	2.62%	2007

4.4.3 Analysis of results

To establish the significance of the results in table 4.11, two-sample t-tests were conducted on the metric describing chess performance. Two tests were performed, comparing the alternate anatomies to the baseline. The verdicts of these tests depend on $\alpha = 0.1$, for a confidence interval of 90%. The results of these tests are listed in table 4.12.

Table 4.12: Statistical comparison between networks of different anatomies

Alternative Hypothesis	t-statistic	p-value	H_{null}
3-layer dense differs from the baseline	-5.9507	0.027	Rejected
3D convolution differs from the baseline	0.2752	0.809	Not rejected

The hypothesis tests show that the dense network is undoubtedly worse than the baseline when considering chess capability. Interestingly, there seems to be no difference in chess prowess between the baseline and the 3D convolutional network.

The chess performance of the 3D network is surprising since it performed better than the baseline in classifying positions, especially when normalizing the output class sizes. It is also interesting to consider that the dense network performed better than the baseline when the validation accuracy was normalized.

These results may provide a hint that the quality of the data used in training these evaluation functions might prove to be a limiting factor in the outright chess prowess of each function.

4.5 Investigating width in dense networks

This section investigates the effect of depth and width in dense networks. This investigation is not done on the basis of anything in the literature, but as a more pragmatic approach to

improve supervised chess engines.

The reasoning behind this pragmatism, is that a lighter network (one with less parameters) can be evaluated faster, which means that the move tree can be evaluated deeper. The move tree refers to the possible future positions that are possible from the current position, given legal chess moves are made. The deeper this tree is evaluated, the less likely it is to make a blunder.

This section in particular aims to have a lighter faster network, that might not be as equipped to evaluated complex positions, but can evaluate much more of them in time controlled chess.

4.5.1 Experimental setup

The networks that will be investigated in this section will all be dense networks, similar to the one featuring in the previous section. Dense networks fared poorly in the previous section, but to exclude the possibility that it was due to architectural choices, a more thorough investigation will be done here. The only difference in these networks will be the number of neurons in each layer. Apart from the 768 wide network in the previous section, three distinct *widths* will be investigated. The networks and their parameters are listed in table 4.13. The specific architectures can be found by amending[1] table 4.9 with the width of the new networks.

Table 4.13: Architectures, width and parameters of dense networks

Architecture	Width	Parameters
Dense	64	61,891
Dense	192	234,307
Dense	384	615,811
Dense	768	1,821,187

It is worth noting that the parameters in different types of networks cannot be compared directly, since convolutional networks share parameters. Sharing of parameters imply that each parameters is used more than once in an evaluation, adding to the computational complexity.

For all the networks in question, classifier accuracy will be measured exactly like elsewhere in this dissertation, but chess performance will be measured slightly differently. In an effort to save time, the majority of the networks' chess performance will be evaluated in chess games where the search depth is limited to 5,000 nodes. The largest and smallest networks will be evaluated in the usual 50,000 depth as well, to be able to compare them to the rest of the chess

[1] Replacing the value in the Output Shape column with the new widths.

in this dissertation.

The chess performance will seem worse when the search depth is limited. This is due to the way Stockfish calibrates[2] its strength when aiming for a specific Elo value [4]. The chess performance of the limited depth chess games can therefore not be compared to the chess performance of the regular depth games without some adjustment.

The hypothesis for this experiments have been revealed in the rationale for this experiment. It is expected that the reduction of width of the networks will have an adverse effect on both chess performance and classifier accuracy.

4.5.2 Results

The networks investigated in this section were all trained the same way, making use of the usual training and validation data breakdown. They were trained across three seeds over five epochs. The results are listed in table 4.14 and 4.15. More detailed results can be seen in Appendix A.2.

Table 4.14: Comparison of dense networks of different widths

Architecture	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
768-wide Dense	58.82%	43.39%	62.23%	62.02%	5.94%	1700
384-wide Dense	59.73%	43.72%	64.51%	62.07%	4.6%	–
192-wide Dense	60.38%	44.48%	61.87%	65.70%	5.87%	–
64-wide Dense	60.30%	44.04%	63.66%	64.01%	4.47%	2232

Table 4.15: Limited depth chess comparison of dense networks of different widths

Architecture	Validation Accuracy	Normalized VA	Limited Depth Elo Rating	Elo Rating
768-wide Dense	58.82%	43.39%	1510	1700
384-wide Dense	59.73%	43.72%	1690	–
192-wide Dense	60.38%	44.48%	1850	–
64-wide Dense	60.30%	44.04%	1945	2232

[2]Stockfish will generate multiple moves, and report back a worse one more often when the performance cap is higher. When the search depth is limited more than normal, the ratio of bad moves go down. This is a product of the pruning in the search tree, and a more in depth explanation is beyond the scope of this research.

4.5.3 Analysis of results

The chess performance of the model improved as width decreased! To assess the significance of this frankly astonishing result, two-sample t-tests were conducted on the chess performance (Elo rating) to assess if there was a difference between the widest and the narrowest of networks. Then, the result of the 64-wide dense network was compared to the baseline to obtain a significance figure to assess how certain one can be that it is as much better as the results claim. The verdict of these tests are based on a confidence interval of 90% ($\alpha = 0.1$), and listed in table 4.16.

Table 4.16: Statistical comparison between dense networks of different widths

Null Hypothesis	t-statistic	p-value	H_{null}
64-wide dense is similiar to the baseline	4.33	0.0495	Rejected
64-wide dense is similiar to 768-wide dense	20.88	0.0023	Rejected

The hypothesis tests confirmed that a reduction in width had a positive effect on chess performance. The positive effect observed was so large, that it was not only better than the baseline, but among the best networks considered in the entire dissertation.

The classifier performance is visually compared to the width of the dense network in figures 4.5 and 4.6.

These figures show an almost linear decline in chess performance as network width increases. Classifier accuracy also shows a downward trend as network width increases. Apart from the fact that a reduction in parameters has caused an increase in chess performance, and to a lesser extent, classifier performance, the narrowest of the models tested performed extremely well.

The 64-wide dense model achieved chess performance comparable with the best classifiers considered throughout this dissertation, and yielding the second best classification performance overall. This poses questions about whether or not convolutional networks contribute anything to performance, and if, when optimized, convolutional and dense networks perform similarly, with dense networks being lighter.

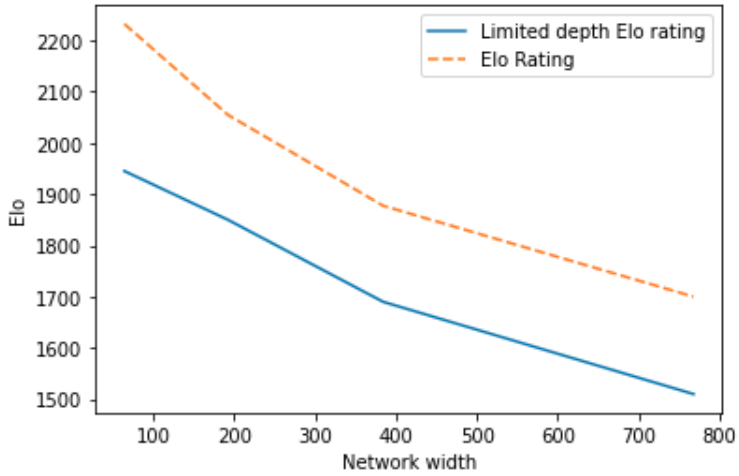

Figure 4.5: Chess performance against dense network width

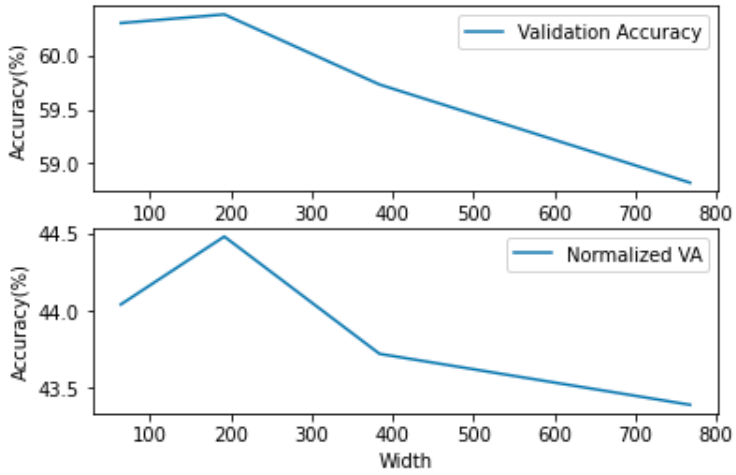

Figure 4.6: Classifier accuracy against dense network width

4.6 Concluding Remarks

This section yielded interesting results, to say the least. Increasing the depth of convolutional networks proved to be a profitable exercise when considering most metrics, confirming the consensus in the literature. Furthermore, neural network chess engines trained with reinforcement learning (like AlphaZero and Leela) use deep convolutional networks, showing that this network topology transcends training methods.

Three dimensional convolutional networks proved to be useful as well, performing on par

with the baseline in terms of chess performance, but slightly better in pure classification terms. While not stated in the results, the resources needed to train and use these 3D networks are high enough that it probably wouldn't be justifiable in practice.

Finally, the dense networks. Initially, the dense networks performed so much worse than the convolutional networks that it seemed that using convolutional networks is a no-brainer. But, as soon as the width of the dense networks declined, both chess and classification performance overtook that of the baseline. This is a result that is in stark contrast with the relevant literature, and practises in the industry, leading to the following recommendation:

Narrow dense networks are much easier to optimize than convolutional networks, and because of their relative lightness, they are more suited to be used in a Minimax chess engines. In contrast, convolutional networks are hard to optimize, expensive to train and expensive to use, but if resources are not limited, the best option for state of the art chess evaluation.

Chapter 5

Balancing under- and overfitting

This chapter will document the investigation of techniques generally used to reduce overfitting. Overfitting is one of the largest detractors of a general neural network classifier performance, which implies that reducing overfitting will increase generalization. These techniques (different activation functions and normalization techniques) will be applied to the baseline neural network, and will be evaluated in terms of chess performance and classifier accuracy.

5.1 Relevant literature

Activation functions are important in neural networks because they introduce non-linearities to the model, which in turn are neccesary to approximate complex functions. In [42], the authors compare various versions of the Rectified Linear Unit in image classification problems. In these comparisons, the Leaky ReLU activation function performed favourably, compared to Parameterized Leaky ReLU (PReLu), and Randomized Leaky ReLU(RReLU). This poses the question of whether or not this performance gain can be observed in chess classification too.

When considering the problem of learning rates, [43] suggests that increasing the batch from which the gradient is computed, will have the same effect as decreasing the learning rate. In addition, using larger batch sizes and decaying the learning rate, proved to be even more advantageous.

Regularization of neural networks have been proven to be useful. Regularization, as defined by [23], is "any modication we make to a learning algorithm that is intended to reduce its generalization error but not its training error". There are many regularization techniques, but arguably the best technique is dropout. Dropout is a technique developed by [44], that is used to mimic a large number of neural networks working on the same problem. By selectively excluding different nodes in a network when updating weights during training, the network as a whole will be less prone to overfit on training data. Work by [45] proposed that dropout for convolution layers can contribute to better classifier performance.

Finally, batch normalization, developed by [46], seeks to reduce the shift in distributions between different layers of a neural network by normalizing the output of each layer. This allows for higher learning rates, but it also desensitizes the network to weight initialization anomalies.

5.2 Activation functions

This section will address the effect that activation functions have on chess performance. Activation functions are important because they are directly responsible for the gradient of the loss function. Collectively, academia has adopted the Rectified Linear Unit (ReLU), but novel activation functions are still being developed. In this section, the ReLU activation used on the baseline will be compared to the older Sigmoid[1] function, and a modified ReLU function known as Leaky ReLU. The definitions of these functions are:

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{5.1}$$

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{5.2}$$

$$f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{5.3}$$

with 5.1 the definition of the Sigmoid, 5.2 of the ReLU, and Leaky ReLU described by 5.3.

[1] The logistic function is used in this section

5.2.1 Experimental setup

For this experiment, the baseline model was taken, and every ReLU was replaced by the activation in question. Table 5.1 shows the network architecture for the sigmoid activations, and is exactly the same as the baseline, and the architecture used for the Leaky ReLU experiment.

For no particular reason, the alpha value for the Leaky ReLU function was chosen to be 0.1 (There is no consensus on a proper way to select the alpha value in the literature, but most publications use values in the order of 0.1, and hence this was the selection made for this experiment).

Table 5.1: Architecture for sigmoid activation functions

Layer (type)	Output Shape	# of Params	Activation
Conv2D	(12, 8, 32)	2336	Sigmoid
Conv2D	(12, 8, 32)	9248	Sigmoid
Conv2D	(12, 8, 32)	9248	Sigmoid
Max Pooling	(6, 4, 32)	0	—
Dropout	(6, 4, 32)	0	—-
Flatten	(768)	0	—
Dense	(64)	49216	Sigmoid
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		70,243 0 70,243	

The development of alternatives to sigmoid based activations came with the adoption of deeper and deeper networks. When networks grow deeper, the values of the outputs can get really large. This posed a problem during training, as the gradient of sigmoid based functions are very small for very large inputs. This is known as the vanishing gradient problem [47]. Vanishing gradients are problematic since the optimization relies specifically on gradients, as in the case in gradient descent.

The fact that the networks considered in this section are not extensively deep leads to the hypothesis that the baseline and its ReLU activation functions will perform poorly when compared to the other networks in this section. This can be attributed to the fact that, by definition, ReLU discards half of the data that it encounters. The same argument leads to the expectation that sigmoids should perform better than Leaky Relu, because there is more detail encoded

in the gradient of the logistic function than in that of the Leaky Relu, which has two constant gradients for positive and negative inputs respectively.

5.2.2 Results

The networks pertaining to this section were trained in exactly the same fashion as the baseline model. Three random seeds were used to train three networks, each over five epochs of the data, using batch sizes of 256 and a cross-entropy loss function. The average results of each seed is shown in table 5.2.

Table 5.2: Comparison of different activation functions

Activation function	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
Sigmoid	59.51%	42.96%	61.92%	64.67%	2.30%	2140
Leaky ReLU	59.51%	43.00%	57.95%	68.46%	2.59%	2061
ReLU (Baseline)	59.45%	42.93%	57.81%	68.34%	2.62%	2007

More detailed results of individual seeds can be viewed in Appendix A.3.

5.2.3 Analysis of results

Given that the classification prowess across all three different networks seemed eerily similar, hypothesis tests was used to show how much off the difference between the chess performances was due to randomness. Two-sample t-tests were conducted to this end, with a verdict centered around a confidence interval of 90% ($\alpha = 0.1$). The tests were performed on the Elo rating achieved by each agent. The results are shown in table 5.3.

Table 5.3: Statistical comparison between different activation functions

Null Hypothesis	t-statistic	p-value	H_{null}
Sigmoid is similar to the baseline	3.0405	0.0933	Rejected
Leaky ReLU is similar to baseline	1.0192	0.4153	Not rejected

The hypothesis tests showed that there was ample evidence to suggest that the sigmoidal based network outperformed the baseline in chess. The same cannot be said about Leaky ReLU, though, with a 40% chance that the observation could just as well have been due to variance, and not a difference in chess playing capabilities.

From the results, it is interesting to note that the difference between the validation accuracies of

all the networks trained for this particular experiment are negligible, but the chess performance is not. Why this particular phenomenon is present is unclear, but it does suggest that one cannot simply optimize these evaluation functions around their classifier performance alone. This also means that training metrics can not necessarily be used to determine when to stop training your model. This will be considered in the next section.

5.3 Optimizations in training procedures

This section will address the effect training procedures has on chess performance, and to a lesser extent classifier accuracy. It is common practice to stop training a model as soon as the validation accuracy tapers of, as an attempt to curb over-fitting the model. But as shown in the previous section, these models' validation accuracy and its chess performances are not inexorably linked. The number of training epochs a network will be trained over will therefore be investigated. Because batch size is a hyperparameter pertaining to training, it was also be under the magnifying glass for this section.

5.3.1 Experimental Setup

The baseline model, and every other model listed in this dissertation, was trained over five epochs. The models used to assess the effect of the number of epochs will be trained over a single epoch (one-shot learning) and ten epochs. Apart from these changes, the models are exactly the same as the baseline.

The models associated with batch size were trained over five epochs each, but during training each of the models' batch size was varied. The various batch sizes along with an identifier are listed in table 5.4. For reference, the baseline was trained using batch sizes of 256.

Table 5.4: List of batch sizes

Designator	Batch size
Batch-128	128
Baseline	256
Batch-512	512
Batch-1024	1024

To hypothesize over training epochs is once again complicated. Five epochs were selected for the baseline, because the baseline training over more epochs did not have an immediate effect

on classifier accuracy. Therefore, the network trained over a single epoch should fare worse than the baseline when considering validation accuracy. The network trained over ten epochs should not yield a discernible difference. There seems to be a reasonably strong correlation between validation accuracy and chess performance (not extremely strong, hence this section), so a safe bet would be that chess performance, when compared to the baseline, should decline in the case of the 1-epoch model and remain unchanged for the 10-epoch case.

A larger batch size generally gives a better indication of the gradient of the loss-landscape, so here a larger batch size should yield a better classifier than the smaller ones. Chess performances should follow a similar trend.

5.3.2 Results

Apart from the changes stipulated in the experimental setup, these models were trained and evaluated exactly as the baseline. The average results of each network seed is shown in table 5.5. More detailed results can be found in Appendix A.3.

Table 5.5: Comparison of training permutations

Designator	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
1-epoch	58.81%	42.72%	60.07%	64.96%	3.15%	1914
10-epoch	59.45%	43.91%	60.7%	65.72%	2.30%	1995
Batch-128	59.16%	42.71%	62.31%	63.43%	2.38%	1921
Batch-512	59.76%	43.01%	60.83%	65.87%	2.53%	2032
Batch-1024	59.88%	43.24%	64.84%	62.04%	2.85%	2033
Baseline	59.45%	42.93%	57.81%	68.34%	2.62%	2007

5.3.3 Analysis of results

This section attempts to distinguish between networks that were trained over many epochs, and few epochs, and between networks that were trained using small batches, and large ones. To this extent, two-sample t-tests was performed to evaluate how significant the differences in chess performances between the these networks are. The verdicts of these tests are evaluated based on a confidence interval of 90%, or a threshold value of $\alpha = 0.1$. The results are listed in table 5.6.

Ignoring statistical significance for a second, it would appear that training for more epochs did

Table 5.6: Statistical comparison between different training permutations

Null Hypothesis	t-statistic	p-value	H_{null}
1-epoch is similar to the baseline	-1.7489	0.2207	Not rejected
1-epoch is similar to the 10-epoch	-1.595	0.2517	Not rejected
10-epoch is similar to the baseline	-0.220	0.8463	Not rejected
batch-128 is similar to the baseline	-2.0881	0.1720	Not rejected
batch-128 is similar to batch-1024	3.9733	0.057	Rejected
batch-1024 is similar to the baseline	0.5324	0.6477	Not rejected

not have an effect on chess performance either. The one-shot learning model did however, fare worse than the baseline.

The hypothesis test showed one significant result. It is almost certain that the smallest batch size network was worse than the largest one, but there is no discernible difference between the largest batch size and the baseline, regardless of the difference in validation accuracy.

Figures 5.1 and 5.2 show a visual representation of how classifier accuracy and chess performance changes with training batch size.

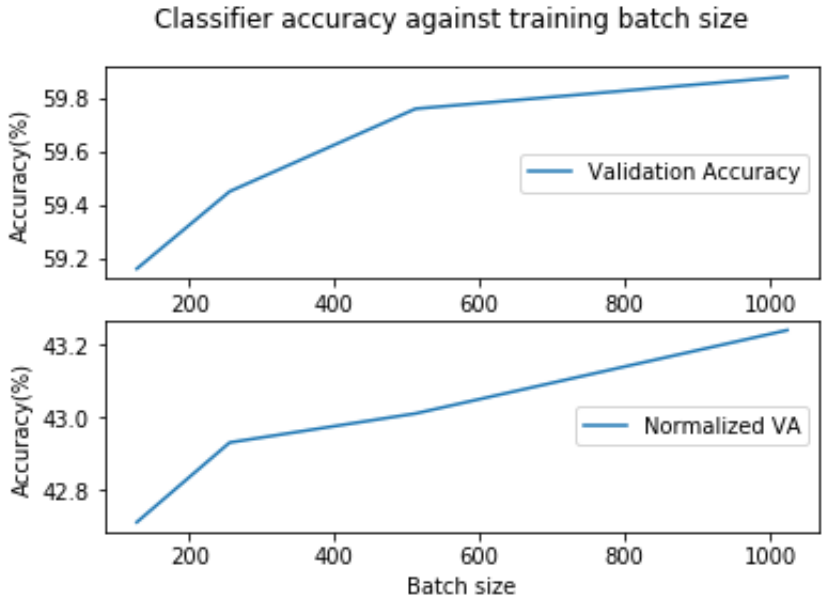

Figure 5.1: Classifier accuracy against training batch size

Contrasting these two figures once again highlights the disparity between accuracy and chess performance. Increasing batch size trains a more general network, but it does not necessarily train a more capable chess evaluation function. By implication then, when striving for the

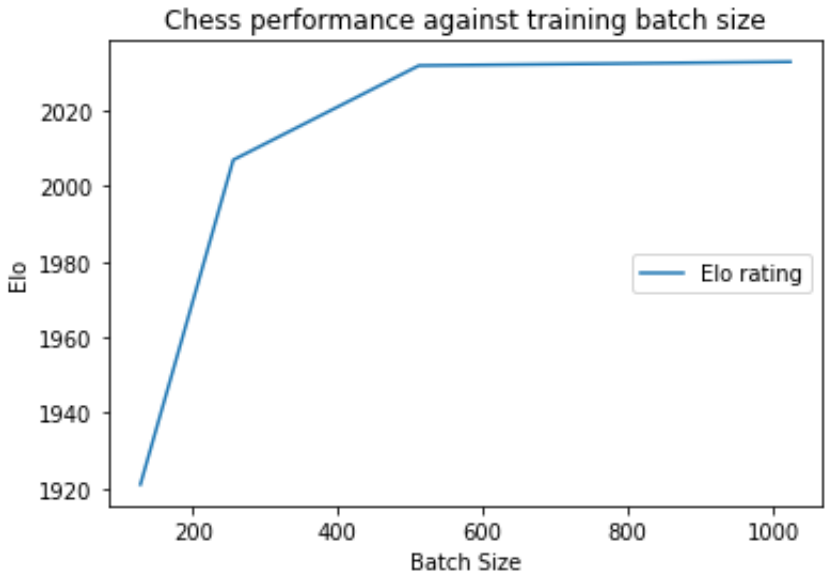

Figure 5.2: Chess performance against training batch size

best chess evaluation function, one cannot only optimize a classifier, as it becomes a frivolous exercise when the gains in classification accuracy diminishes.

5.4 Batch normalization and dropout

This section will investigate the effect of batch normalization and dropout on both chess performance and classifier accuracy. Batch normalization is the process of taking a set of data, subtracting the mean of the data, and dividing by the standard deviation, while dropout refers to the probabilistic deactivation of network parameters during training.

5.4.1 Experimental Setup

The effect batch normalization was investigated by introducing batch normalization after every layer of the baseline. This introduces a few additional (untrainable) parameters to the network, thus the architecture is presented in table 5.7.

The literature makes mention of some adverse effects when applying dropout to convolution layers [48], so the effect of dropout on convolution layers in a chess evaluation function was gauged by removing the dropout layer after the convolution layers of the baseline, but still keeping it after the dense layers. The effect of dropout in general will be measured by com-

pletely removing dropout from the baseline. For reference, the probability of dropout is 0.5% throughout all the networks.

Table 5.7: Architecture of batch normalized network

Layer (type)	Output Shape	# of Params	Activation
Conv2D	(12, 8, 32)	2336	ReLU
Batch Norm	(12, 8, 32)	128	–
Conv2D	(12, 8, 32)	9248	ReLU
Batch Norm	(12, 8, 32)	128	–
Conv2D	(12, 8, 32)	9248	ReLU
Batch Norm	(12, 8, 32)	128	–
Max Pooling	(6, 4, 32)	0	—
Dropout	(6, 4, 32)	0	—-
Flatten	(768)	0	—
Dense	(64)	49216	ReLU
Batch Norm	(12, 8, 32)	128	–
Dropout	(128)	0	—
Dense	(3)	195	Softmax
Trainable params Non-trainable params Total params		70,563 320 70,883	

The effect of dropout is said to diminish as the size of the training set increases, and since the training set is quite large, the application of dropout, on both dense and convolution layers was expected to have little impact on classifier accuracy.

Batch normalization is really valuable when training very deep networks or using sigmoidal activation functions, neither of which the baseline does. Due to the removal of covariate shift[2], batch normalized networks are less sensitive to changes in learning rate, and while none of the networks trained use fixed learning rates, the adaptive learning rates are more likely to yield a better model in conjunction with batch normalization. The batch normalized networks should therefore perform slightly better with regards to classifier performance, but a gain in chess performance seems unlikely.

5.4.2 Results

The networks under investigation in this section are trained exactly the same as the baseline, consisting of three random seeds, trained over five epochs. The averaged results of each seed

[2]When the distribution of the independent variables (the inputs to the networks) changes.

is shown in table 5.8. In this table, some dropout refers to the network that only had dropout applied after each dense layer.

Table 5.8: Comparison of batch normalized and dropout networks

Designator	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
Some dropout	59.57%	43.23%	58.81%	67.51%	3.38%	2073
No dropout	59.78%	43.78%	60.18%	66.34%	4.81%	2031
Batch normalized	60.22%	43.58%	65.74%	61.97%	3.05%	2212
Baseline	59.45%	42.93%	57.81%	68.34%	2.62%	2007

A more detailed set of results for each individual seed can be found in appendix A.3.

5.4.3 Analysis of results

Hypothesis testing was conducted to ascertain the significance of the chess performance results in the previous section. These hypothesis tests were two-sample t-tests (on the chess performance metric), and their verdict revolved around a threshold value of $\alpha = 0.1$, for a confidence interval of 90%. The results are listed in table 5.9.

Table 5.9: Statistical comparison between regularization techniques

Null Hypothesis	t-statistic	p-value	H_{null}
Some dropout is similar to the baseline	1.2449	0.3393	Not rejected
No dropout is similar to the baseline	0.5584	0.6327	Not rejected
Batch normalization is similar to the baseline	4.6036	0.0440	Rejected

The hypothesis tests showed that batch normalization has a statistically significant performance gain over the baseline in chess performance. It also performed exceptionally well in classification, suggesting that the baseline architecture is sensitive to changes in learning rate.

The dropout networks both showed an increase in classifier accuracy and chess performance. Using no dropout yielded, in comparison to the other dropout cases, the highest classification accuracy, but not the best chess performance. This further solidifies the proposal that validation accuracy and chess performance is strongly, but not perfectly correlated.

5.5 Concluding remarks

This section compared various techniques that strive to optimally fit a model to a certain problem. Activation functions did not have a noticeable effect on traditional classification performance, but sigmoids showed a clear improvement in specifically chess performance.

This lead to the rejection of the assumption that validation accuracy is the optimal way to gauge when to stop training, training the model longer than initially believed to be sufficient bore no fruit. Fiddling with batch sizes proved to increase classification performance, but was found wanting when considering chess performance.

When considering regularization techniques, test results showed that dropout had a detrimental effect on classifier accuracy, suggesting that the dataset is varied enough to not take advantage of the benefits of dropout. Chess performance varied slightly, but statistical analysis showed that this was really insignificant.

Batch normalization proved to be invaluable, leading to –relative to other techniques considered– massive gains in both classification and chess prowess.

Chapter 6

Conclusion

This study sets out to compare various neural network generalization techniques when applying them to a chess evaluation function that was obtained by means of supervised learning. As a side effect, it documented the process of optimizing a classifier for something ever so slightly different than classification.

Because the state of the art neural network chess evaluation functions are very expensive to train, a new simpler evaluation model was developed. This evaluation function was inspired by an ideal evaluation function, that could in theory map any given chess position to the eventual outcome of the game. Since a chess match can end in three different states, the new evaluation function had three outputs, which in hindsight was a mistake. This can be attributed to two main factors. The largest contributor is that when the evaluation function is used by a chess engine, moves are chosen which reduces a player's probability of losing, or increases a player's probability of winning. Whether or not a game ends in a draw is completely ignored. The other factor is that when considering all the available training data, the amount that ends in a draw is almost negligible. This has the implication that even when using the evaluation function as a classifier, the draw class is almost entirely useless, as can be confirmed by the dismal performance that all the classifiers showed on that specific class.

Nevertheless, the research question that this dissertation needs to answer is whether or not generalization techniques can improve chess performance of a supervised evaluation function. This is of interest because there is no doubt that generalization techniques can improve a neural

network based classifier, as confirmed by the results presented in this dissertation. Table 6.1 shows a summary of the best performing classifier from each chapter, showing that every area investigated yielded positive results.

Table 6.1: Summary of the best performing classifiers

Chapter	Network	Validation Accuracy	Elo Rating
2	Baseline Seed #1	**59.87**%	2034
2	Baseline Seed #3	59.24%	**2066**
3	1x8x8 Embedding Seed #3	**60.53**%	2210
3	2x8x8 Embedding Seed #3	60.30%	**2318**
4	64 Wide Dense Seed #2	**60.53**%	2192
4	64 Wide Dense Seed #1	60.14%	**2304**
5	Batch Normalized Seed #1	**60.40**%	2189
5	Batch Normalized Seed #2	60.15%	**2253**

Actual chess performance provided slightly different results. Of the 102 evaluation functions that were experimented on, the following metrics were measured:

- Training accuracy
- Validation accuracy
- Normalized validation accuracy
- Use case accuracy
- Win accuracy
- Lose accuracy
- Draw accuracy
- Leading Accuracy
- Losing Accuracy

The correlation between these accuracies and chess performance is listed in table 6.2. A few interesting observations can be made from this table. There exists a strong correlation between validation accuracy and chess performance. Validation accuracy therefore, is a good indicator

of how well a specific classifier will fare in chess. It is not, however the best indication. The use case accuracy metric is the best metric to optimize for when trying to achieve the best chess playing evaluation function.

Another interesting observation is that every class accuracy has a negative correlation with chess performance, except for the ”lose” class. This might be an artifact of evaluating every position from white's perspective, but it can also be an indication that knowing which chess moves to avoid serves a player better than knowing which moves to make.

Table 6.2: Correlation between classification and chess performance

Classification metric	Correlation between metric and chess performance
Training accuracy	-0.07
Validation accuracy	0.73
Normalized validation accuracy	0.35
Use case accuracy	0.74
Win accuracy	-0.23
Lose accuracy	0.39
Draw accuracy	-0.19
Leading Accuracy	-0.25
Losing Accuracy	0.38

A final observation that stems from these correlations is the fact that while the correlation is strong, ($r \approx 0.75$), there is no perfect correlation. This means that an increase in the above mentioned metrics might even result in a poorer chess elevation function. As consequence, the best evaluation function won't be found by looking at these metrics, chess games will have to be played to establish the very best from the best.

A possible explanation for the imperfect correlation can be assigned to the quality of the chess games the classifier was trained on. The training data was sampled randomly from the Lichess database, meaning that games of any skill level could have been selected, and a classifier is only as good as the data it was trained on.

To answer the research question though, it must be concluded that applying generalization techniques on any area of the network will have a positive effect on the chess performance of a supervised chess evaluation function. In addition to this improvement, generalization techniques also eases the process of optimizing your classifier, which leads to even better chess results.

Finally, a word on supervised chess evaluation functions in general. Search depth has an enormous impact on the ability of a chess engine to play properly. Feeding data through large neural networks are expensive. So expensive in fact, that when used in conjunction with Alpha-Beta based MiniMax, it almost certainly is not worth it to use them. When considering supervised learning in conjunction with a probabilistic tree search like MCTS, the reduction in training time on it's own causes it to be a lucrative endeavor.

References

[1] 2020. [Online]. Available: https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

[2] J. Russell and J. Russell, "Google ai beats go world champion again to complete historic 4-1 series victory," Mar 2016. [Online]. Available: https://techcrunch.com/2016/03/15/google-ai-beats-go-world-champion-again-to-complete-historic-4-1-series-victory/?_ga=2.176483808.416731520.1552117150-415071195.1552117150

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: http://science.sciencemag.org/content/362/6419/1140

[4] Stockfish, "Latest version of stockfish," https://github.com/official-stockfish/Stockfish, 2019.

[5] 2019. [Online]. Available: https://tcec.chessdom.com/

[6] Leela, "Latest version of leelachesszero," https://github.com/glinscott/leela-chess, 2019.

[7] G. Haworth and N. Hernandex, 2019. [Online]. Available: https://tcec.chessdom.com/articles/TCEC_15_for_TCEC.pdf

[8] "the new neural network based engine entering tcec s15," 2019. [Online]. Available: http://www.chessdom.com/alliestein-the-new-neural-network-entering-tcec-s15/

[9] A. Silver, "Fat fritz what on earth is that?" 2019. [Online]. Available: https://en.chessbase.com/post/fat-fritz-what-on-earth-is-that

[10] 2019. [Online]. Available: https://en.chessbase.com/post/fat-fritz-defeats-stockfish-match-2

[11] C. E. Shannon, *Programming a Computer for Playing Chess*. New York, NY: Springer New York, 1988, pp. 2–13. [Online]. Available: https://doi.org/10.1007/978-1-4757-1968-0_1

[12] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[13] S. Thrun, "Learning to play the game of chess," in *Advances in Neural Information Processing Systems 7*. The MIT Press, 1995, pp. 1069–1076.

[14] J. Baxter, A. Tridgell, and L. Weaver, "Learning to play chess using temporal differences," *Machine Learning*, vol. 40, no. 3, pp. 243–263, Sep 2000. [Online]. Available: https://doi.org/10.1023/A:1007634325138

[15] J. Veness, D. Silver, A. Blair, and W. Uther, "Bootstrapping from game tree search," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 1937–1945. [Online]. Available: http://papers.nips.cc/paper/3722-bootstrapping-from-game-tree-search.pdf

[16] M. Lai, "Giraffe: Using deep reinforcement learning to play chess," *CoRR*, vol. abs/1509.01549, 2015. [Online]. Available: http://arxiv.org/abs/1509.01549

[17] E. David, N. Netanyahu, and L. Wolf, "Deepchess: End-to-end deep neural network for automatic learning in chess," 09 2016, pp. 88–96.

[18] 2019. [Online]. Available: https://chess-db.com/public/top100alltime.jsp

[19] L. Wu, Z. Zhu, and W. E, "Towards understanding generalization of deep learning: Perspective of loss landscapes," 2017.

[20] A. Zhou, Y. Ma, Y. Li, X. Zhang, and P. Luo, "Towards improving generalization of deep networks via consistent normalization," 2019.

[21] D. H, "How much did alphago zero cost?" 2018. [Online]. Available: https://www.yuzeh.com/data/agz-cost.html

[22] C. M. BISHOP, *PATTERN RECOGNITION AND MACHINE LEARNING*. SPRINGER-VERLAG NEW YORK, 2016.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[24] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[25] 2019. [Online]. Available: https://keras.io/examples/mnist_cnn/

[26] 2019. [Online]. Available: https://www.chessprogramming.org/UCI

[27] 2019. [Online]. Available: https://ccrl.chessdom.com/ccrl/404/

[28] 2020. [Online]. Available: https://database.lichess.org/

[29] M. Jaritz, R. Charette, E. Wirbel, X. Perrotton, and F. Nashashibi, "Sparse and dense data with cnns: Depth completion and semantic segmentation," 09 2018, pp. 52–60.

[30] E. David, N. Netanyahu, and L. Wolf, "Deepchess: End-to-end deep neural network for automatic learning in chess," 09 2016, pp. 88–96.

[31] J. Brownlee, "Why one-hot encode data in machine learning?" 2019. [Online]. Available: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

[32] J. Capablanca and N. de Firmian, *Chess Fundamentals (Completely Revised and Updated for the 21st century)*. Random House Puzzles Games, 2006.

[33] G. Foody, M. Mcculloch, and W. Yates, "The effect of training set size and composition on artificial neural network classification," *International Journal of Remote Sensing - INT J REMOTE SENS*, vol. 16, pp. 1707–1723, 06 1995.

[34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[35] S. Loussaief and A. Abdelkrim, "Convolutional neural network hyper-parameters optimization based on genetic algorithms," *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, vol. 9, no. 10, pp. 252–266, 2018.

[36] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," 2015.

[37] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 6231–6239. [Online]. Available: http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf

[38] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 922–928.

[39] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, Jan 2013.

[40] P. Molchanov, S. Gupta, K. Kim, and J. Kautz, "Hand gesture recognition with 3d convolutional neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2015.

[41] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[42] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," 2015.

[43] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," 2017.

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[45] H. Wu and X. Gu, "Towards dropout training for convolutional neural networks," *Neural networks : the official journal of the International Neural Network Society*, vol. 71, pp. 1–10, 07 2015.

[46] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[47] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[48] H. Jansma, "Don't use dropout in convolutional networks - kdnuggets," 2020. [Online]. Available: https://www.kdnuggets.com/2018/09/dropout-convolutional-networks.html

Appendix A

Seed specific results

In this appendix, the results of the individual seeds of the neural network classifiers are documented. As a reminder, in the respective chapters the results of these seeds were averaged for the sake of brevity.

A.1 Manual input encodings, embeddings and datasets

Table A.1: Results of the dense model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.988%	43.957%	58.410%	68.740%	4.720%	2154
B	59.829%	44.137%	63.650%	63.490%	5.270%	2051
C	60.116%	44.413%	65.230%	61.490%	6.520%	2151

Table A.2: Results of the sparse model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	58.801%	42.230%	76.290%	48.820%	1.580%	1878
B	59.260%	42.350%	63.990%	62.110%	0.950%	2014
C	59.355%	42.667%	66.740%	59.260%	2.000%	1889

Table A.3: Results of the 8x8x2 Embedding model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	60.303%	44.123%	56.630%	70.870%	4.870%	2318
B	60.516%	43.573%	64.210%	63.830%	2.680%	2221
C	60.416%	43.870%	59.900%	68.170%	3.540%	2169

Table A.4: Results of the 8x8x1 Embedding model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.965%	43.840%	55.280%	71.540%	4.700%	2242
B	60.464%	43.703%	63.310%	64.500%	3.300%	2264
C	60.525%	43.867%	64.130%	63.850%	3.620%	2210

Table A.5: Results of the 8x8x2 Manual model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.014%	42.043%	64.100%	60.900%	1.130%	2002
B	58.491%	41.633%	44.530%	79.360%	1.010%	2035
C	58.826%	42.237%	61.700%	62.980%	2.030%	1934

Table A.6: Results of the 8x8x1 Manual model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	58.546%	42.303%	58.540%	65.720%	2.650%	1949
B	58.684%	42.017%	69.610%	54.860%	1.580%	1956
C	58.909%	42.543%	53.480%	71.390%	2.760%	2049

Table A.7: Results of the Less chess model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.460%	42.827%	54.050%	71.810%	2.620%	1993
B	59.501%	43.093%	56.520%	69.560%	3.200%	1906
C	59.059%	42.513%	62.270%	63.290%	1.980%	1853

Table A.8: Results of the Even less chess model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.100%	42.390%	63.380%	62.160%	1.630%	1908
B	59.194%	43.087%	65.720%	60.010%	3.530%	1875
C	59.378%	43.617%	58.690%	66.600%	5.560%	1954

Table A.9: Results of the Almost no chess model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	58.842%	41.943%	63.600%	61.040%	1.190%	1718
B	58.527%	42.733%	63.570%	60.240%	4.390%	1698
C	54.981%	43.083%	70.420%	44.880%	13.950%	1619

Table A.10: Results of the No Draw model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.314%	42.133%	55.320%	71.080%	0.000%	2066
B	59.617%	42.240%	63.140%	63.580%	0.000%	2009
C	59.226%	42.050%	73.410%	52.740%	0.000%	1949

A.2 Network anatomy and topology

Table A.11: Results of the 1 layer Convolutional model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.465%	42.417%	57.900%	68.000%	1.350%	1937
B	59.181%	42.607%	61.300%	64.170%	2.350%	1932
C	58.964%	42.407%	58.040%	67.420%	1.760%	1863

Table A.12: Results of the 2 layer Convolutional model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.497%	43.037%	53.030%	72.870%	3.210%	1996
B	59.074%	42.500%	74.400%	51.230%	1.870%	1824
C	59.538%	43.097%	67.400%	58.690%	3.200%	1920

Table A.13: Results of the 4 layer Convolutional model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.337%	43.140%	54.420%	71.500%	3.500%	2041
B	59.660%	43.203%	62.110%	64.580%	2.920%	2007
C	59.155%	43.063%	62.670%	63.250%	3.270%	2044

Table A.14: Results of the 5 layer Convolutional model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.438%	43.300%	51.330%	74.080%	4.490%	2089
B	59.129%	43.827%	64.690%	60.690%	6.100%	2054
C	59.860%	43.870%	59.280%	67.190%	5.140%	2160

Table A.15: Results of the 8 wide Convolutional model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.149%	42.770%	72.730%	52.800%	2.780%	1960
B	59.200%	42.690%	62.450%	63.040%	2.580%	1916
C	58.804%	42.017%	53.440%	71.500%	1.110%	1865

Table A.16: Results of the 16 wide Convolutional model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.054%	42.653%	70.090%	55.580%	2.290%	1906
B	59.714%	43.407%	58.020%	68.370%	3.830%	1984
C	59.218%	42.500%	58.760%	67.100%	1.640%	1919

Table A.17: Results of the 48 wide Convolutional model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.696%	43.327%	66.750%	59.640%	3.590%	1894
B	59.642%	43.340%	68.060%	58.320%	3.640%	1939
C	59.391%	43.520%	64.470%	61.380%	4.710%	1916

Table A.18: Results of the 64 wide Convolutional model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.545%	43.300%	61.110%	65.440%	3.350%	1950
B	58.563%	42.603%	76.640%	48.480%	2.690%	1910
C	59.285%	43.157%	65.820%	60.180%	3.470%	1920

Table A.19: Results of the 3-layer Dense model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	58.804%	43.417%	61.180%	63.140%	5.930%	1745
B	59.038%	42.857%	58.820%	66.080%	3.670%	1715
C	58.619%	43.920%	66.690%	56.840%	8.230%	1635

Table A.20: Results of the 3D-Convolutional model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.870%	43.140%	63.040%	64.030%	2.350%	2014
B	59.702%	43.133%	55.330%	71.210%	2.860%	2052
C	59.945%	43.677%	62.850%	63.740%	4.440%	1995

Table A.21: Results of the 64-wide Dense model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	60.141%	43.720%	65.280%	62.210%	3.670%	2304
B	60.528%	44.363%	61.270%	66.540%	5.280%	2192
C	60.241%	44.053%	64.420%	63.290%	4.450%	2218

Table A.22: Results of the 384-wide Dense limited depth model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.441%	43.697%	66.760%	59.620%	4.710%	1687
B	59.638%	43.417%	68.680%	57.840%	3.730%	1679
C	60.130%	44.067%	58.090%	68.750%	5.360%	1706

Table A.23: Results of the 192-wide Dense limited depth model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	60.430%	44.333%	62.680%	65.130%	5.190%	1858
B	60.152%	44.720%	62.130%	65.110%	6.920%	1831
C	60.555%	44.387%	60.800%	66.850%	5.510%	1866

A.3 Balancing under- and overfitting

Table A.24: Results of the Sigmoid model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.538%	42.717%	56.170%	70.610%	1.370%	2136
B	59.343%	43.103%	65.840%	60.150%	3.320%	2114
C	59.648%	43.067%	63.750%	63.250%	2.200%	2177

Table A.25: Results of the Leaky ReLU model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.586%	43.073%	61.560%	65.030%	2.630%	2018
B	59.647%	42.730%	59.740%	67.020%	1.430%	2044
C	59.306%	43.210%	52.570%	73.340%	3.720%	2134

Table A.26: Results of the 1-epoch model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.199%	43.210%	58.630%	66.930%	4.070%	1944
B	58.687%	42.533%	46.330%	78.240%	3.030%	1957
C	58.569%	42.440%	75.260%	49.720%	2.340%	1849

Table A.27: Results of the 10-epoch model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.434%	42.673%	60.210%	66.440%	1.370%	2072
B	59.841%	43.487%	58.550%	68.080%	3.830%	1973
C	59.082%	42.563%	63.340%	62.640%	1.710%	1952

Table A.28: Results of the Batch-128 model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.031%	42.517%	53.390%	72.170%	1.990%	1936
B	59.187%	42.677%	67.460%	58.330%	2.240%	1905
C	59.283%	42.927%	66.080%	59.780%	2.920%	1924

Table A.29: Results of the Batch-512 model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.808%	43.357%	70.040%	56.600%	3.430%	2023
B	59.838%	43.167%	56.650%	70.230%	2.620%	2109
C	59.636%	42.710%	55.790%	70.800%	1.540%	1979

Table A.30: Results of the Batch-1024 model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.962%	43.283%	67.530%	59.380%	2.940%	1985
B	59.950%	43.370%	66.950%	60.160%	3.000%	2074
C	59.736%	43.080%	60.060%	66.580%	2.600%	2046

Table A.31: Results of the Some dropout model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.516%	43.453%	62.960%	63.200%	4.200%	2135
B	59.798%	43.350%	57.580%	69.110%	3.360%	2082
C	59.414%	42.893%	55.890%	70.210%	2.580%	2015

Table A.32: Results of the No dropout model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	59.653%	44.420%	59.710%	66.260%	7.290%	2008
B	59.883%	43.310%	62.960%	64.030%	2.940%	2033
C	59.804%	43.603%	57.870%	68.730%	4.210%	2053

Table A.33: Results of the Batch normalized model

Seed	Validation Accuracy	Normalized VA	Win Accuracy	Loss Accuracy	Draw Accuracy	Elo Rating
A	60.404%	43.580%	66.190%	61.470%	3.080%	2189
B	60.150%	43.470%	64.550%	63.260%	2.600%	2253
C	60.132%	43.683%	66.480%	61.100%	3.470%	2200