

MSc Dissertation Project

# *Transfer Learning and Moonboard Climbing*

*Name: Thomas Hughes*

*ID: B6027480*

*Supervisor: Chris Bates*



*This dissertation does **NOT** contain confidential material and thus can be made available to staff and students via the library*

## **Abstract**

Within recent years, the sport of rock climbing has surged in popularity, with many climbers seeking tools to increase their strength and technique. One such tool is the Moonboard, a standardised climbing wall with a global app, allowing climbers to set and attempt problems, established by other users. However, these grades often lack accuracy, hindering training and leaving climbers questioning their own performance.

Previous research has attempted to address the arbitrariness within grading on the Moonboard by applying various supervised learning techniques, however, they often faced issues surrounding overfitting and bias. Despite this, improvements have been made through increased optimisation of features.

One significant limitation in current research is its lack of foresight. The models produced are restricted to the specific Moonboard set on which they were trained, rendering them quickly outdated as new Moonboard sets are continually introduced

This project aims to explore the application of transfer learning across multiple Moonboard sets by developing a predictive model capable of accurately grading climbing problems. This endeavour not only enhances the climber's experience but also lays the groundwork for future research, mitigating issues related to redundancy.

## **Acknowledgements**

I would personally like to thank my supervisor Chris Bates for his continued support and guidance throughout the project. Also, the Depot Climbing Centre; without access to the Moonboard, labelling the datasets would have been near impossible. Finally, I would like to thank my girlfriend, Isabelle, for her continued support.

# Contents

<b>Abstract.....</b>	<b>1</b>
Acknowledgements.....	2
<b>Introduction.....</b>	<b>5</b>
<b>1 Research Question, Aims &amp; Objectives.....</b>	<b>6</b>
1.1 Research Question.....	6
1.2 Objectives.....	6
1.3 Deliverable.....	7
<b>2 Literature review.....</b>	<b>7</b>
2.1 Moonboard.....	7
2.2 Grading Climbing Problems.....	8
2.2 Previous Work.....	9
2.2.1 Convolutional Neural Networks.....	9
2.2.2 BetaMove.....	10
2.2.3 Heuristic Approach.....	11
2.2.4 Graph Theory.....	12
2.2.4 Artificial Creativity.....	13
2.3 Conclusion.....	13
<b>3 Research Design.....</b>	<b>13</b>
3.1 Philosophy.....	14
3.2 Approach.....	14
3.3 Methodology.....	14
<b>4 Design and Implementation.....</b>	<b>15</b>
4.1 Software Tools.....	15
4.2 Data Collection and Preprocessing.....	16
4.2.1 Data Collection.....	16
4.2.2 Initial Preprocessing.....	18
4.2.3 Sequencer.....	18
4.2.4 Recurrent Neural Network - Features.....	24
4.2.5 Recurrent Neural Network - One Hot encoding.....	24
4.2.6 Recurrent Neural Network - Padding.....	25
4.2.7 Recurrent Neural Network - Splitting Training & Validation.....	25
4.2.8 Recurrent Neural Network - Masking Layer.....	25
4.2.9 Recurrent Neural Network - Long Term Short Term Memory (LSTM).....	26
4.2.10 Recurrent Neural Network - Softmax.....	27
<b>5 Evaluation.....</b>	<b>28</b>
5.1.1 2016 Training Set.....	29
5.1.2 2017 Training Set.....	30
5.1.3 Mixed Training Set.....	31
5.2 Further Analysis.....	33
5.2.1 Under-Grading.....	33
5.2.2 Training Set Impact.....	34

5.3 Conclusion.....	35
<b>6 Limitations and Further Work.....</b>	<b>36</b>
6.1 The Sequencer.....	36
6.2 Hold Types.....	37
6.3 Future Work.....	37
6.3.1 Alternatives to Recurrent Neural Networks and LTSM.....	37
6.3.2 Alternate Features.....	37
6.3.3 Real Rock and Other Standardised Boards.....	38
<b>7 Pragmatism &amp; gradePredictor.....</b>	<b>38</b>
7.1 gradePredictor.....	38
7.2 Limitations.....	39
7.3 Conclusion.....	39
<b>References.....</b>	<b>41</b>

## Introduction

Rock climbing has become increasingly popular, leading many climbers to seek out training tools that can help them improve their skills and strength. One such tool is the Moonboard, a standardised climbing wall that allows climbers from around the world to set problems via the Moonboard app for others to try and climb. Each problem is assigned a grade by the user to indicate its difficulty. However, since these grades are arbitrarily assigned, they can often be inaccurate and negatively impact the climbers' training.

To address this issue, researchers have attempted to classify the difficulty of problems using various supervised learning techniques such as Naive Bayes, Softmax Regression, and Neural Networks (*A. Dobles et al, 2017*). However, overfitting and bias within the dataset have limited the success of these models and produced poor results in terms of accuracy. Despite these challenges, further optimisation and variations in representing the dataset have led to improvements (*R. Chang & Y. Duh, 2021*).

While improvements have been made, current literature only explores the option of producing models capable of accurately predicting on a single Moonboard set. However, this leads to a form of redundancy as new Moonboard sets are constantly being updated and released; as of writing this (2023), three Moonboard sets are available. This redundancy hinders progress and limits the scope of available research.

By aiming to produce a model capable of accurately predicting across multiple Moonboard sets, we not only minimise future redundancies, but also establish a framework for further work to be carried out within the domain of *transfer learning*\* and climbing.

\* *Transfer learning is a machine learning technique where knowledge gained from training on one task is applied to improve performance on a different but related task.*

# 1 Research Question, Aims & Objectives

The aim of this research is to produce a model capable of accurately predicting the grade of climbing *problems*\*, irrespective of which Moonboard set they are climbed on. This will ultimately improve the climbers experience, as they can more accurately track their improvements without worrying whether the grade assigned truly represents the difficulty.

*\* Climbing "problem" refers to a specific climbing route or sequence of holds on a climbing wall or rock face that climbers attempt to solve and ascend. Note - When the word problem is referring to a climbing problem, it will be italicised*

## 1.1 Research Question

*Can a machine learning model be developed to consistently and accurately predict climbing problem grades, achieving consistent performance across various Moonboard sets, and thereby enhancing the climbing experience for users seeking grade consistency and reliability?*

## 1.2 Objectives

The following table outlines the criteria for successful research to be undertaken:

Objective	Measure
To review current literature in the context of the Moonboard and machine learning, identifying any existing gaps of knowledge or limitations.	A complete literature review, concluding with current limitations found in the existing research.
Collect and preprocess data from multiple Moonboard sets.	A dataset will be produced with <i>problems</i> from each of the existing Moonboard sets.
Transform data such that it can be processed by the algorithm to build a model from	The model will not encounter any errors/issues surrounding the input of data
Develop some form of <i>machine learning</i> algorithm capable of accurately predicting across multiple sets.	Accuracy scores will be produced to evaluate the capability of successfully predicting when faced with data from multiple Moonboard sets.
Identify limitations within the project and suggest possible solutions and further work to be carried out.	Limitations will be presented with possible solutions, while also producing a section that covers future work

## 1.3 Deliverable

With pragmatism in mind, a model will be produced and made publicly available via my Github repository. This model should be capable of grading problems regardless of the Moonboard set they climbed on. Although, the main deliverable of this work is perhaps more abstract, being to further the understanding of *transfer learning* within climbing and provide future researchers with a framework to build upon.

## 2 Literature review

The following literature review aims to explore current research and literature surrounding the Moonboard, climbing and machine learning, while also providing more context for those unfamiliar with climbing and its grading systems.

### 2.1 Moonboard

The Moonboard is a standardised training tool for climbers, created by Ben Moon in 2004 (released commercially in 2016) to train for hard climbing moves in a compact and consistent way. Its grading system ranges from 6A+ to 8B+ (V3 to V14), based on the difficulty of moves and intensity of the climb. It is widely used in climbing gyms as a benchmark for climbers' abilities and as a training tool to develop grip strength over various hold types (*J. Medernach et al, n.d.*). The grading system is constantly evolving based on the consensus of experienced climbers who have tried the problems.



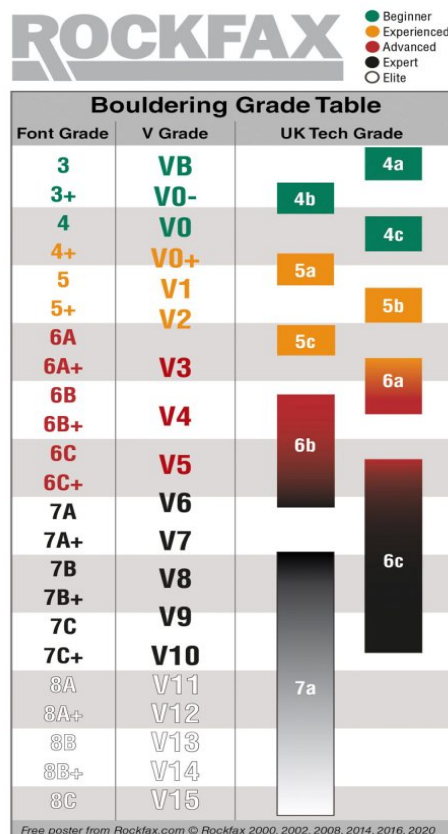
Figure 1: A climber (myself) on the Moonboard



## 2.2 Grading Climbing Problems

A climbing *problem* refers to a particular route on either a rock face or climbing wall. The route will usually have a designated start and finish point, with the climber either matching both hands to the final hold or *topping out*\*. Between the start and the finish, there will be a number of holds the climber must use to navigate to the top; which holds and the sequence the climber chooses often determines the success of the climb, however discovering the correct sequence is often difficult and hence the route is referred to as a *problem*.

The grading of climbing *problems* has traditionally been closely tied to geographic regions. For instance, in Europe, including the UK, the widely accepted grading system is the French *Font* grading scale. Meanwhile, in North America, climbers predominantly utilise the Vermine Grade (*V-Grade*). Although other more esoteric grading systems exist, like Japan's *Dankyu* system, the global climbing community primarily relies on the Font and V-grades. Both the V-Grade and Font Scale are similar in that they describe the overall difficulty of a climb, as opposed to other systems such as the now defunct UK Tech grade which measures the difficulty of the single hardest move on the problem. Furthermore, they both scale mostly linearly to each other, as such they are often used interchangeably.



**ROCKFAX**

● Beginner  
● Experienced  
● Advanced  
● Expert  
○ Elite

Font Grade	V Grade	UK Tech Grade
3	VB	4a
3+	V0-	4b
4	V0	4c
4+	V0+	5a
5	V1	5b
5+	V2	5c
6A	V3	6a
6A+	V4	6b
6B	V5	6c
6B+	V6	
6C	V7	
6C+	V8	
7A	V9	
7A+	V10	
7B	V11	
7B+	V12	
7C	V13	
7C+	V14	
8A	V15	
8A+		
8B		
8B+		
8C		

Free poster from Rockfax.com © Rockfax 2000, 2002, 2008, 2014, 2016, 2020

Figure 2: Boulder grades conversion table

\* 'Topping out' refers to climbing over the top of the rock face

Image source: [www.rockfax.com](http://www.rockfax.com)

## 2.2 Previous Work

The following section provides an overview of previous work carried out within the domain of climbing, with particular emphasis on the use of machine learning algorithms.

### 2.2.1 Convolutional Neural Networks

In their research, *A. Dobles et al. (2017)* utilised machine learning techniques to estimate the grade of boulder problems. They specifically utilised the *Convolutional Neural Network* (CNN) technique, which is commonly used for pattern recognition in images to identify objects. The researchers represented the Moonboard routes using a multi-hot encoded format, where each route is depicted as a 140-dimensional vector, where each dimension corresponds to the existence or nonexistence of one of the 140 holds. This representation only includes information about which holds are present in the route and which are not, as opposed to more implicit features such as distance and hold type. Unfortunately the classifier performed poorly on unseen data - predicting with an accuracy score of **36.5%**. Comparatively, human performance when measured by allowing climbers to climb the route and then predict the grade, resulted in an average accuracy score of **93.4%**. This poor performance was attributed partly to the uneven distribution of problems throughout the grades, with there being far more grades at the lower end of the spectrum. However, they also suggested increasing the feature space, such as rating the difficulty of each hold, as a possible future endeavour.

### 2.2.2 BetaMove

In their research, *Y.S. Duh & R. Chang (2021)* again utilised neural networks to assess and generate climbing route difficulty. They developed three separate neural network models: BetaMove, GradeNet, and DeepRouteSet. BetaMove was created to estimate the sequence of moves within the climb, GradeNet estimated climbing route difficulty while taking into account the sequence provided by BetaMove, and DeepRouteSet produced new climbing routes.

They chose to use a *Recurrent Neural Network* (RNN) algorithm as one can not only handle variable length inputs, but also has the ability to maintain memory of past states, which allows for the model to capture *temporal dependencies*\*. The ability to capture *temporal dependencies* allows the model to excel in areas where the sequence of input matters, such as climbing. For instance, multiple difficult moves in a row would almost certainly be more challenging than one hard move, followed by an easy move, then by another hard move etc.

When evaluating the model on the training set, they were able to achieve an accuracy rate of **64.3%**, however, the model's performance dropped fairly significantly when tested against *unseen*\* data to **46.7%** - suggesting possible overfitting within the model. Although, this **46.7%** figure still exceeded the human level performance of **45%** that they determined via a

survey, where climbers would predict the grade of a climb by only observing the problem (as opposed to climbing the problem, as in 2.2.1). They also reported an accuracy rate to *plus or minus one* grade; this resulted in far greater performance against the test set, measuring **84.8%**, only slightly below the human level performance of **87.5%**. The increase in performance may be attributed to nuances between grades that aren't appropriately captured by the model.

The quality of the routes generated by DeepRouteSet was evaluated through a survey by experienced climbers, which assessed four categories: unnecessary holds, strange moves/fluency of moves, reasonable route, and route quality. When presented together with Moonboard routes, DeepRouteSet was found to be more successful in all four categories. The researchers believe that the success of GradeNet and DeepRouteSet is due to the success of BetaMove, which has successfully produced and evaluated routes suitable for climbing with the betas provided.

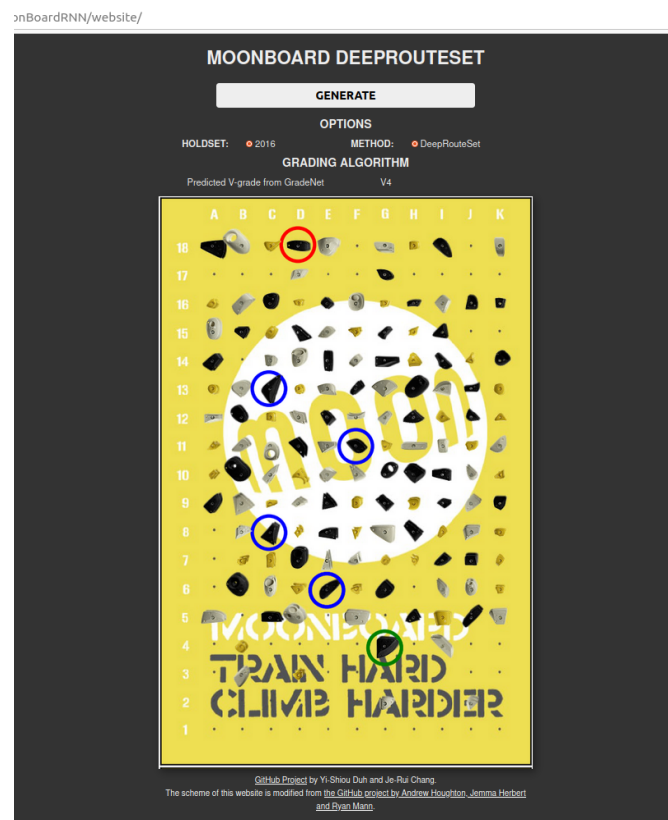


Figure 3: Problem generated by DeepRouteSet

\* 'Temporal dependencies' refer to the sequential relationships and order of events or data points over time, where one event's occurrence or value depends on or influences those preceding it.

\* 'Unseen' data refers to data that has not been used within the training phase

Image source: <https://jrchang612.github.io/MoonBoardRNN/website/>

### 2.2.3 Heuristic Approach

In their study, *F. Stapel (2020)* employed a *heuristic\** approach to create novel climbing routes. This approach involved classifying climbing holds and moves and assessing their influence on route difficulty, with a specific focus on Moonboard problems. The study operated on the premise of a correlation between hold difficulty and route grade. To gauge hold difficulty, a survey was administered, with climbers assigning ratings on a scale ranging from one ("An easy hold; easily gripped with one hand") to five ("A challenging hold; primarily suitable as an intermediate hold").

---

**Algorithm 1** A greedy algorithm for route generation  
**Data:** Desired difficulty, hold types and length modifier  
**Result:** A list of coordinates forming a climbing route  
route = []  
  **while** last hold is not in row 18 **do**  
    calculate the score of each hold  
    add best scoring hold to route

---

Figure 4: Pseudocode for Greedy algorithm implemented

Subsequently, a Greedy algorithm (see **Figure X**) was employed to identify and select holds with the highest scores. The selection criteria encompassed hold type, suggested difficulty levels, as well as considerations related to distance and rotational angle within the model's context. The resultant routes were evaluated via a questionnaire based on their grade, overall *flow* (how natural the movements feel), and climbers' enjoyment.

However, it was observed that the generated routes exhibited less favourable *flow* compared to existing Moonboard routes. This drawback was attributed to the algorithm's failure to consider the climber's last position when introducing a new hold to the sequence. Additionally, the algorithm did not account for footholds, presenting another limitation in its route generation capabilities.

*\* A heuristic approach is a problem-solving or decision-making strategy that uses a practical approach or rule of thumb to find a satisfactory solution, often relying on simplifications or educated guesses.*

### 2.2.4 Graph Theory

*D. Seal & R. Seal (2022)* incorporated graph theory into their research on climbing and routing. Their work was inspired by training boards like the Moonboard, and they created a simulated artificial climbing wall with holds placed randomly. They then selected a starting and ending hold from these randomly placed holds. Using *Dijkstra's* algorithm, they determined the shortest path of holds between the start and end points, which are suitable for

climbers with specific physical characteristics. They defined adjacent nodes and edges based on these physical properties, and created graphs from the appropriate nodes after filtering out the unsuitable holds. Their results showed that the model found different routes for climbers of different heights. They compared the difficulty of routes based on the number of nodes and total distances, where longer routes with more nodes are considered more difficult. They recommend incorporating hold properties into the difficulty estimation of routes.

## 2.2.4 Artificial Creativity

Inspired by *A. Dobles et al, H. Brand (2019)* explored the domain of ‘Artificial Creativity’. They implemented both a *Generative Adversarial Network* (GAN) and a *Variational Autoencoder* (VAE). However, as their project progressed the performance of the GAN did not improve as expected; this was thought to be due to inherent instabilities in GANs, such as *Mode Collapse*\*. As such, they concentrated their efforts on implementing the VAE. In order to assess the generated routes they created a survey on the ‘/r/climbharder’ subreddit, asking experienced climbers to evaluate the problem on several attributes; unfortunately they received too little responses to provide any significant result.

*\* Mode collapse in machine learning refers to a situation where a generative model, like a GAN (Generative Adversarial Network), produces limited or repetitive output, failing to capture the full diversity of the data it is trained on.*

## 2.3 Conclusion

When confronted with the existing literature, a number of limitations and gaps can be identified, they include, but are not limited to:

- **Performance on Unseen Data:** Several of the studies faced challenges when applying their models to unseen data, indicating potential issues with model generalisation and overfitting.
- **Subjective Evaluation:** Many studies relied on surveys or human evaluations to assess the quality of generated climbing routes, which can introduce subjectivity and bias into the assessments.
- **Failure to Capture Climbing Nuances:** Many studies struggled to capture the subtleties of climbing difficulty, especially when it comes to nuances between different grades or the *flow* of climbing movements.
- **Limited Feature Space:** The representation of climbing routes or holds was often simplified, lacking more comprehensive information about features such as hold type or distance

While these limitations present inherent challenges for the project, addressing and mitigating them will be instrumental in achieving improved and more robust outcomes.

### 3 Research Design

The following section describes the research philosophy, approach and methodology to be followed throughout the project

#### 3.1 Philosophy

The research philosophy being employed may be considered somewhat *pragmatic\**, as the scope of the research involves producing a model (D.L. Morgan, 2014) which is beneficial to climbers looking for consistency and accuracy within their climbing. However, it could also be argued *positivism\** is more appropriate, as the research will focus on identifying optimal features and maximising the effectiveness of the models predictions, thus suggesting a greater emphasis on objective and empirical methods (P. Ralph, 2018) for testing and verifying a hypothesis. Furthermore, the emphasis on the transfer learning side suggests a focus on developing a more generalised knowledge base that can be applied beyond the context of the project (G. Guba, 1990).

*\*A pragmatic research philosophy emphasises practicality, focusing on using methods and approaches that yield actionable and real-world solutions to research problems.*

*\* Positivism is a research philosophy that emphasises the objective and empirical study of phenomena, relying on systematic observation and measurement to generate knowledge grounded in observable facts and causal relationships.*

#### 3.2 Approach

Without a defined hypothesis, *inductive reasoning\** aligns well with the project's objectives. The primary goal is to construct a theoretical framework based on empirical data, enabling the generation of new knowledge and insights in the field of climbing problem grade prediction. *Inductive reasoning* facilitates the exploration of patterns, relationships, and emerging principles within the data, allowing for the development of a robust foundation for the study's findings and the potential identification of novel perspectives and approaches

*\* Inductive reasoning is a logical process of drawing general conclusions or principles from specific observations or evidence, making it a method for generating broad hypotheses based on empirical data.*

#### 3.3 Methodology

A *quantitative\** research methodology is well-suited to the project, as the aim is to empirically investigate the feasibility of developing a machine learning model that consistently and accurately predicts climbing problem grades across various Moonboard sets.

While there may not be a specific hypothesis, the research question guides towards systematically collecting and analysing numerical data to assess the model's performance in achieving grade consistency and reliability. This approach aligns with established practices in quantitative research and is supported by both J.W. Cresswell (2014) and E. Babbie (2016), who emphasise the value of quantitative methods for data-driven inquiries and empirical investigations.

*\* Quantitative research is an empirical approach that involves collecting and analysing numerical data to discover patterns, relationships, and trends and to test hypotheses or research questions.*

## 4 Design and Implementation

The following section describes the tools and techniques used to implement the software, as well as the reasoning and justifications behind the usage of certain libraries, models and techniques.

### 4.1 Software Tools

With a tremendous amount of modules and libraries available, *Python* was the obvious choice for the implementation of not only the machine learning aspect of the project, but the preprocessing also. The following libraries were used throughout:

**Numpy:** An essential library that efficiently handles the processing of numerical data. It was used to organise and manipulate data before the training phase of the machine learning model carried out with *TensorFlow*.

**TensorFlow:** A deep learning framework that enables the creation and training of neural network models. It was used to define, train, and evaluate the machine learning model, leveraging its powerful capabilities for neural network development and optimisation.

**Scikit-learn (sklearn):** A crucial machine learning library that provides a wide range of tools and algorithms for tasks such as classification, regression, clustering, dimensionality reduction, and more. It was used for tasks such as data preprocessing, splitting datasets for training and testing, and evaluating the machine learning model's performance, making it an indispensable part of the machine learning workflow.

**Matplotlib:** A fundamental data visualisation library in Python that is essential for creating a wide range of plots and charts. It was used to visually represent and communicate research findings, enabling the clear and informative presentation of data and results.

**Seaborn:** A powerful data visualisation library in Python that builds on Matplotlib and simplifies the creation of aesthetically pleasing and informative statistical graphics. It was used to enhance the visual representation of data (heatmaps), making it easier to explore patterns and relationships within the dataset.

**Pandas:** A data manipulation and analysis library in Python that provides data structures (e.g., DataFrames and Series) and tools for efficiently handling and exploring structured data. It was used for tasks such as data loading, cleaning, transformation, and exploration, playing a crucial role in preparing and understanding the data.

## 4.2 Data Collection and Preprocessing

The following section will discuss how initially the data was collected, as well as the necessary steps taken to prepare and transform the data to a format suitable for machine learning.

### 4.2.1 Data Collection

The majority of previous researchers had implemented web scrapers in order to collect the dataset of climbs. A popular example being Github user *gestalt-howard*'s moonGen, which cycled through the list of problems determining which holds were present given the image of the climb. Unfortunately however, these web scrapers are now all but defunct due to changes in the Moonboard website, which restricts access to the problem list. Now the only option that exists would involve scraping the mobile app; a much more laborious task as all the information would have to be determined from screenshots alone. This route was explored before an alternative solution became apparent. The alternative came in the form of a *Kaggle* dataset, from user Edward Minks (<https://www.kaggle.com/datasets/eddous/moonboard>). This publicly available dataset was near perfect. It contained problems from each year's Moonboard set in the form of a *JSON* structure, as shown in *Figure 5*.



```

"Method" : string "Feet follow hands"
"Name" : string "BYE BYE BYE"
"Grade" : string "7A+"
"FirstName" : string "Flo"
"LastName" : string "Brand"
"IsBenchmark" : bool false
"IsMaster" : bool false
"IsAssessmentProblem" : bool false
"MoonBoardHoldSetup" : string "MoonBoard Masters 2019"
"MoonboardConfiguration" : string "40° MoonBoard"
"RepeatText" : string "Be the first to repeat this problem"
"DateInserted" : string "2020-02-23T21:08:37.397Z"
"DateTimeString" : string "23 Feb 2020 21:08"
"NameForUrl" : string "bye-bye-bye"
"Moves" : [ 5 items
  0 : { 3 items
    "Position" : string "H4"
    "IsStart" : bool true
    "IsEnd" : bool false
  }
]

```

*Figure 5: Example of JSON structure*

However, there were several notable limitations to the dataset. The most significant challenge was the lack of data for the 2019 Moonboard setup. While the 2016 and 2017 datasets contained over 20,000 data points each, the 2019 dataset had fewer than 2,000. This disparity could greatly affect the training and performance of the models, and as such it was decided to exclude the 2019 climbs from the final datasets. Having two datasets, although not ideal, would still allow for an evaluation of transfer learning. Finally, It's also important to acknowledge that while Kaggle provided a valuable resource, the reliability of external datasets, including those on Kaggle, can vary. Therefore, precautions were taken to validate and preprocess the data.

### 4.2.2 Initial Preprocessing

From the initial JSON structure to a format appropriate for training, a number of preprocessing steps were necessary. The following outlines the steps taken.

#### Step 1: Converting from JSON to CSV

- The decision was made to convert the dataset from the original JSON format to a more readable CSV file. This aided *data consistency* by reducing potential issues caused by irregular or nested JSON data.

#### Step 2: Exclusion of 2019 Dataset

- As mentioned previously, the dataset originally contained data for the 2016, 2017, and 2019 Moonboard setups. Given the substantial difference in data volume, the decision was made to exclude the 2019 dataset from further analysis. This choice aimed to maintain dataset consistency and prevent the potential skewing of model training and performance due to data imbalance.

#### Step 3: Removing Non-Repeats

- After successfully completing a climb, each user is given the opportunity to vote on the grade which they perceived the problem to be. User voting on perceived grades enhances dataset quality by incorporating *collective wisdom*, fostering grade consensus and assuring data quality. The removal of climbs with zero repeats was necessary to prioritise climbs for which there was a level of consensus among climbers regarding their difficulty and grade, ensuring more reliable grade assignments.

#### Step 4: Converting Font to V-Grade

- For the most part, Font and V-grades scale linearly with each other. However, the exception lies at the lower end of the grade spectrum, with Font opting for a higher level of granularity. For example, V5 in the V-grade scale could convert to either 6C or 6C+. By converting to V-grades, the data sparsity was reduced, resulting in a more balanced distribution of climbing problems across the grade range. The reduction in scope also enhances generalisation, as the model is less likely to overfit to minor variations in difficulty within a grade. Finally, converting to V-grades helps simplify the problem, creating a simpler and more manageable target.

### 4.2.3 Sequencer

The heuristic approach taken by *F. Stapel (2020)*, identified two major features for determining difficulty within a climbing problem. The first being the *distance* between holds. The greater the distance between the *current* hold and the *target* hold, the more force the climber would need to generate. Furthermore, if the movement would result in the climbers feet from leaving the footholds (*dyno*), even more contraction strength would be required to hold on (*L.V. Giles et al, 2006*). The second feature identified was the *hold type/size*. With smaller holds providing less surface area to distribute the climbers weight and grip force

effectively, thus increasing the pressure on the climbers fingers (R. Bourne, 2011). Secondly, smaller holds require greater precision when placing the hands; this demands exceptional accuracy and control in hand positioning.

The initial preprocessing (4.2.2) would develop a more robust dataset, however, it would not be possible to determine either the distance between holds or the hold types without further injections and manipulation. The information about holds would be stored in the *Moves* column of the dataset. Each instance would contain a nested list of dictionaries, where each dictionary coincided with a hold that was present in the climb. The *key* attributes are *Position*, *IsStart* and *IsEnd*. For example,

```
[{Position: 'I8', IsStart: False, IsEnd: False},
 {Position: 'D9', IsStart: False, IsEnd: False},
 {Position: 'F5', IsStart: True, IsEnd: False},
 {Position: 'A16', IsStart: False, IsEnd: False}
 {Position: 'D18', IsStart: False, IsEnd: True}
 {Position: 'E13', IsStart: False, IsEnd: False}]
```

The first step would be to establish the hold types. Past researchers (R. Chang & Y. Duh, 2020) had assigned difficulty ratings to each hold on the board. These scores were assigned by experienced climbers, with more difficult to hold positions being assigned higher scores. With this system, each unique hold on the board would have a difficulty associated with it. This technique, although effective, would not translate well within the given research domain of *Transfer Learning*, as the model would become entirely fitted to the specific board set up it was trained upon. Instead, a more generalised approach would be necessary. For this reason, each hold on both the 2016 and 2017 set up would be assigned a *hold type* from a set list. The list would consist of 12 unique categories:

**["Small Crimp", "Medium Crimp", "Large Crimp", "Pocket", "Small Pocket", "Jug", "Small Pinch", "Medium Pinch", "Large Pinch", "Slopy Pinch", "Sloper"]**

The decision making when assigning hold types was mainly arbitrary due to constraints such as the inability to measure holds without removing them from the wall.

After assigning a type to each hold, the information was transposed into the *moves* category within the dataset. Which would now take the form of:

```
[{Position: 'I8', IsStart: False, IsEnd: False, HoldType: Jug},
 {Position: 'D9', IsStart: False, IsEnd: False, HoldType: Jug},
 {Position: 'F5', IsStart: True, IsEnd: False, HoldType: Large Pinch},
 {Position: 'A16', IsStart: False, IsEnd: False, HoldType: Small Crimp}
 {Position: 'D18', IsStart: False, IsEnd: True, HoldType: Large Crimp}
 {Position: 'E13', IsStart: False, IsEnd: False, HoldType: Jug}]
```

After the laborious task of assigning hold types, the next step would be to determine the distance between holds. At first this would appear to be a trivial task; converting the positions to coordinates and taking the euclidean distance between the two holds. However, one issue would become apparent rather quickly. How can we determine the distance between holds when the current *moves* attribute contains nothing about the order in which the holds are to be taken, apart from the fact that one of many holds are the start and finish. For this, a method to determine the sequence would be required. Unlike injecting hold types to the dataset, it would be inconceivable to manually convert each move into the correct sequence; instead a *sequencer* would need to be developed.

The *Sequencer* would in fact be the most difficult and arduous process to implement. Unfortunately, an existing sequencer, such as that developed by *R. Chang & Y. Duh (2020)* would be incompatible as it would only predict the sequence effectively on the 2016 set; whereas sequences for both the 2016 **and** 2017 set would be required. Although not compatible, the *Beam Search* algorithm they implemented would serve as inspiration. Although, as opposed to a *Beam Search* algorithm, an alternative *Greedy* algorithm was opted for instead. This was partly due to the desire to reduce complexity in the developing aspect, but also to reduce processing time.

To implement the *Greedy* algorithm, each hold type as determined previously would be assigned a difficulty rating.

```
difficulties = {  
    "Small Crimp": 9.0,  
    "Medium Crimp": 7.5,  
    "Large Crimp": 6.5,  
    "Pocket": 5.0,  
    "Small Pocket": 9.0,  
    "Jug": 3.0,  
    "Small Pinch": 8.0,  
    "Medium Pinch": 6.5,  
    "Large Pinch": 4.0,  
    "Slopy Pinch": 7.0,  
    "Sloper": 7.5  
}
```

*Figure 6: Difficulty ratings*

These difficulty ratings would be one part to determining which hold would be taken next in the sequence. The other determining factor being the distance between the holds. To determine the distance between the holds, as described earlier, the hold position (e.g. I8, B9, C13 etc) was converted to a coordinate, for example A0 would become 0,0; these coordinates would then be used to determine the *euclidean distance* between holds.

```

def calcDist(self,hold_A,hold_B):
    ''' Calculates the distance between holds (Euclidian distance) '''
    # This may need changing to centimetres if decide to go down outdoor route
    holdA = self.getCoords(hold_A)
    holdB = self.getCoords(hold_B)
    squared_distances = [(p2 - p1) ** 2 for p1, p2 in zip(holdA, holdB)]
    sum_squared_distances = sum(squared_distances)
    distance = math.sqrt(sum_squared_distances)
    return distance

def getCoords(self, hold):
    ''' Converts A1 to 1,1 etc '''
    # Convert letter to column number
    x = ord(hold[0]) - ord('A') + 1
    # Convert number part to row number
    y = int(hold[1:])
    return (x,y)

```

Figure 7: Distance calculations

Before writing the *Greedy* algorithm, a number of precautions would need to be taken to ensure unnatural movements would not occur. The first would prevent the *sequencer* from attempting to move more than the arm span of the climber. This would also prevent the sequencer from continuously moving just one hand. For this, it was assumed that most climbers could not reach more than 10 units.

The second movement that would be assigned a negative weight would be *continuous crossovers*. The most natural movement when climbing a wall would consist of the right hand staying to the right of the left hand and *vice versa*. This movement is often referred to as *ladder climbing*. However, it's not unusual for one hand to cross over the opposing hand, such that the right hand would be to the left of the right hand etc. A single crossover is to be expected, but continuous crossings of the hands would suggest the problem is being climbed incorrectly (or the climb is just poorly set). Therefore, if the hands crossed over simultaneously more than once, the method would recursively call itself, this time with a different starting hand movement.

One of the more complex procedures would be determining which holds are designed for use as footholds only. However, this was simplified under the assumption that footholds would generally be further laterally away from the hand holds and/or under the current hand holds, for instance, **J3** in Figure 8.

As such, if the hold was below either of the current hand holds, it would be assigned a negative weight.

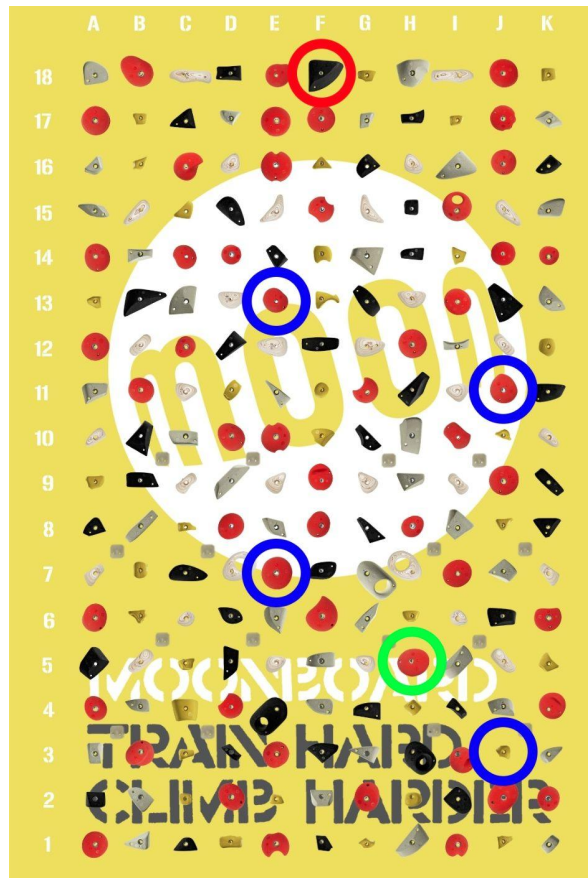


Figure 8: Example of foothold

The final movement that would be punished by the *sequencer* would be *bumps*. A *bump* is where the hand moves more than once sequentially. This decision was somewhat controversial as *bumps* do occur in natural climbing, especially on the more complex/difficult problems. However, without this inclusion, the *sequencer* would continuously *bump* the one hand until the *max span* was reached. By minimising *bumping* the more natural *ladder* movement would be maintained.

These unnatural movements, along with the distance and difficulty scores would be used to determine the hand movements/sequence. The *IsStart* key determined which hold(s) would be assigned as the initial current holds (*currHolds*). Then each available hold would be assigned a weight determined by an addition of the hold type score (Figure 6) and the distance between the two holds. Distance was assigned a multiplier of **1.25**, as it became apparent it was more of a determining factor than hold type. The weights would then be increased if they satisfied any of the unnatural movement conditions. With each move being assigned a weight, the *Greedy* algorithm would choose the lowest score (considered the easiest movement).

```

while True:
    # Lowest value (Assigned arbitrary large number)
    minStep = [100]
    # if lower than current holds remove from usableHolds
    for hold in usableHolds:
        if self.isBelow(currHolds['RH'], list(hold.keys())[0]) or self.isBelow(currHolds['LH'], list(hold.keys())[1]):
            usableHolds.remove(hold)

    for hold in usableHolds:
        holdLoc = list(hold.keys())[0]
        holdType = hold[holdLoc]
        # Get values for both hands
        for hand in currHolds:
            currHoldLoc = currHolds[hand]
            # Calculate the distance between holds
            distance = self.calcDist(currHoldLoc, holdLoc)
            # Evaluate the difficulty of each move based on the reach
            diff = self.evalDifficulty(holdType, distance)
            # Check if hand moved previously
            if hand == handMoved:
                diff += HandMovePenalty
            # Check if the number of cross overs were exceeded previously
            if step == 0 and hand == firstHand:
                # Only applies to first step
                diff += HandMovePenalty
                crossOverExceeded = False
            # Ensure max span has not been exceeded
            if self.calcDist(currHolds['RH'], holdLoc) >= self.maxReach or self.calcDist(currHolds['LH'], holdLoc) >= self.maxReach:
                diff += HandMovePenalty
            # Punish downward movements as these are most likely used
            if self.isBelow(currHoldLoc, holdLoc):
                diff += HandMovePenalty
            #print(diff)
            # Update difficulty
            if diff < minStep[0]:
                minStep = [diff, hand, list(hold.keys())[0]]

    # Remove the selected hold from the usable holds
    usableHolds = [d for d in usableHolds if minStep[2] not in d]

```

Figure 9: Greedy algorithm

To assess the performance of the *sequencer*, a manual labelling process was required for a subset of problems. Specifically, twenty problems were randomly selected from the dataset to form the test set. After testing, an accuracy score of **85%** (17/20) was achieved. Given the relatively small size of the test set, it raises concerns about whether this accuracy score accurately represents the entire dataset. However, considering the limited alternatives available, it was decided to proceed with applying the function to the entire dataset

Ultimately, the output to the *Greedy* algorithm would consist of a list of dictionaries, where the *key* would be the hand and the *value* the hold position, e.g.

{'RH': 'K4'}, {'LH': 'G2'}, {'LH': 'I8'}, {'RH': 'F12'}, {'LH': 'G11'}, {'RH': 'C14'},  
{'LH': 'E16'}, {'RH': 'D18'}}

#### 4.2.4 Recurrent Neural Network - Features

With the availability of sequential data, a *Recurrent Neural Network* (RNN) was opted for based on the promising results reported by R. *Chang and Y. Duh* (2020) in a similar context. However, like all machine learning algorithms, the RNN requires specific input features.

The sequence generated by the sequencer (4.2.3) alone wouldn't suffice. Firstly, it lacks information about hold types, providing only positioning data. To ensure the model's generalisation across Moonboard sets, data would need to be augmented with hold type information. Secondly, distance between holds, a crucial determinant of climbing difficulty (*F. Stapel, 2020*), was missing from the sequence.

Fundamentally, all climbing problems can be thought of as a series of subproblems or moves. In the real climbing world, a move can range from subtle hand readjustments to the more significant act of moving a hand from one hold to another. In our simplified context, each move consists of three essential components: the two initial holds and the target hold. What distinguishes one move from another is the distance between the initial (moving) hold and the target hold, which significantly influences the move's difficulty.

With this in mind, the decision was made to transform the sequencer's output into a structured set of moves. These moves are represented as tuples (*right hand, left hand, target hold, distance*) and populate the *Input* column within the dataset. For example:

[{'RH': 'K4'}, {'LH': 'G2'}, {'LH': 'I8'}, {'RH': 'F12'}, {'LH': 'G11'}, {'RH': 'C14'},  
{'LH': 'E16'}, {'RH': 'D18'}]

$\Leftrightarrow$

[('Large Pinch', 'Large Pinch', 'Jug', 4.24), ('Large Pinch', 'Jug', 'Jug', 4.47), ('Jug',  
'Jug', 'Jug', 6.40), ('Jug', 'Jug', 'Small Crimp', 7.61), ('Jug', 'Small Crimp', 'Large  
Crimp', 5.09)]

#### 4.2.5 Recurrent Neural Network - One Hot encoding

With each climbing problem now represented as a series of subproblems, it became essential to convert the hold types into a format compatible with the RNN. The RNN requires a numerical representation of categorical data, such as the various hold types. To achieve this, *one-hot encoding* was implemented. In this encoding scheme, each hold type is mapped to an 11-dimensional vector, where each dimension corresponds to one of the following hold types: Small Crimp, Medium Crimp, Large Crimp, Pocket, Small Pocket, Jug, Small Pinch,



Medium Pinch, Large Pinch, Slopy Pinch, and Sloper. Only one element in the vector is set to **one** (indicating the presence of that specific hold type), while the others are set to **zero**. For instance, if a climbing sequence involves a 'Small Crimp' hold, the corresponding element in the one-hot encoded vector for 'Small Crimp' is set to **one**, while the remaining dimensions remain **zero**. This encoding ensures that the RNN can effectively process and learn from the categorical data of hold types.

#### 4.2.6 Recurrent Neural Network - Padding

With the Neural Net requiring fixed length inputs, procedures would need to take place to ensure each problem had the same number of moves. The solution to this required *padding* the input sequence with a variable number of trailing zeros, such that a common sequence length was achieved. This was implemented by the *Tensorflow* module *pad\_sequences*.

#### 4.2.7 Recurrent Neural Network - Splitting Training & Validation

To ensure the RNN's ability to generalise effectively while mitigating the risk of *overfitting*, a conventional practice is adopted: the dataset is segregated into two distinct subsets, namely the training set and the validation set. The training set serves as the primary data source for model training, whereas the validation set functions as an independent and unseen dataset for evaluating the model's performance during the training process.

To implement this partitioning, the *train\_test\_split* function from the *sklearn* library was utilised. The common convention entails allocating a specific proportion of the dataset, typically 20%, to the validation set, with the remaining 80% constituting the training set. This distribution ensured an equilibrium between an ample volume of data for training and the capability to effectively assess the model's performance.

The division enables continual monitoring of the model's predictive accuracy and loss metrics on data that it has not been exposed to during training. This process plays a pivotal role in the identification of potential overfitting concerns and guides adjustments in model architecture and training parameters to optimise performance and generalisation.

#### 4.2.8 Recurrent Neural Network - Masking Layer

When defining the model, the first layer after the input to be initialised is the *masking* layer. This layer treats the additional values inserted during the *padding* phase (4.2.6) as missing values, thus ensuring the model does not consider the *padded* values as meaningful input.

## 4.2.9 Recurrent Neural Network - Long Term Short Term Memory (LSTM)

After initialising the *masking* layer (4.2.8), the *recurrent* aspect of the neural net would be implemented. To capture long-range/temporal dependencies and context within the climbing sequences, a Long Short-Term Memory (LSTM) layer is introduced (Hochreiter & Schmidhuber, 1997).

The LSTM architecture's internal memory state enables the network to take into account prior moves when assessing the difficulty of subsequent moves. This capability facilitates the modelling of both short-term and long-term dependencies, although it's worth noting that long-term dependencies may be of lesser significance in the context of climbing problems, which tend to involve relatively short sequences of moves.

Selecting the appropriate number of nodes for the LSTM layer involved an experimental approach aimed at achieving robust generalisation. An unseen dataset was introduced to evaluate the model's performance in terms of accuracy. However, both the *training* and *validation* accuracy scores were considered, with a focus on identifying any significant divergence between them, which could indicate overfitting. Furthermore, a confusion matrix would be introduced to ensure the model was predicting well throughout the grade ranges.

Eventually, after exploring a node count ranging from 8 to 256, an optimal value of 128 was determined. This value would effectively capture the complexity of the problem, without being overly fitted.

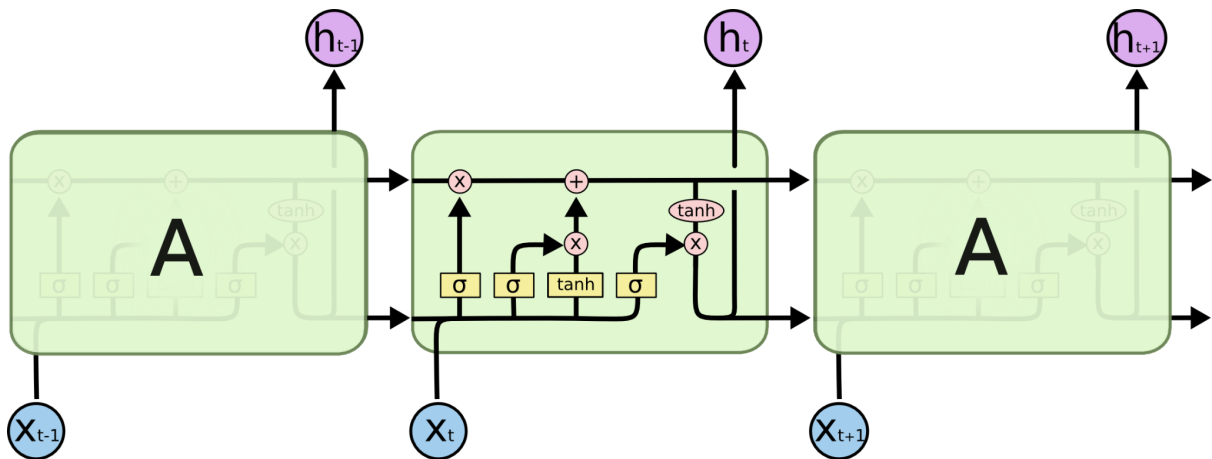
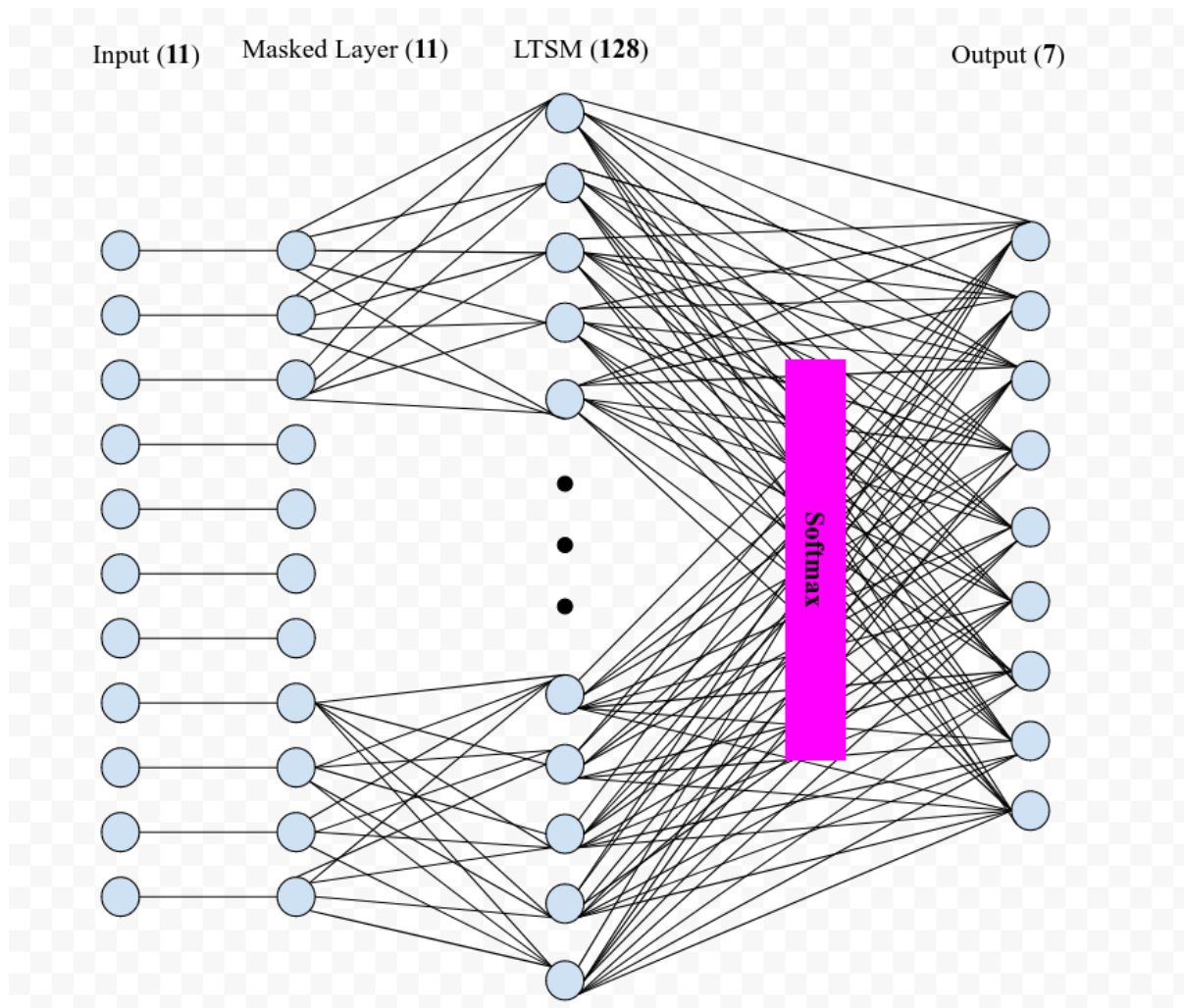


Figure 10: Example of LSTM layer

Image source: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

#### 4.2.10 Recurrent Neural Network - Softmax

Following the introduction of a single LSTM layer, the network is concluded with a final dense output layer. This dense layer consists of 7 nodes, with each node corresponding to a distinct grade category, spanning from V4 to V10. The addition of a softmax activation function to this layer serves to transform the network's raw outputs into a probability distribution across these grade categories. In this distribution, the grade category with the highest probability is selected as the model's prediction.



*Figure 11: Final network architecture*

## 5 Evaluation

When evaluating the performance of the model, it would be imperative to consider the initial objective of this research; that being, to train and produce a model capable of performing well on multiple Moonboard sets. However, the term “performing well” is entirely subjective. Therefore, a more consistent and quantifiable measure would have to be defined. For this, Human-Level Performance\* (**HLP**) would be introduced as the benchmark criteria for which every model would be evaluated against. In addition to evaluating the accuracy of the models, a confusion matrix would be produced for each model in order to assess the capability of predicting accurately across the spectrum of grades.

Three models in total would be produced, each trained on varying datasets. The first would be trained on the 2016 set, the second would be trained on the 2017 set and the third would be trained on a mixture of both sets. Subsequently, each of these models will undergo evaluation using corresponding *test* sets derived from their respective training datasets. These *test* sets would be completely unseen by the training set in order to avoid any *data leakage*\*.

*\* HLP, as defined by R. Chang and Y. Duh (2020), is quantified as achieving an accuracy of 45% for an exact grade match and 87.5% for predictions falling within one grade of the target. This was determined on the 2016 Moonboard set.*

*\* Data leakage occurs when information from outside the training dataset unintentionally influences a machine learning model's performance, leading to over-optimistic results and reduced generalisability.*

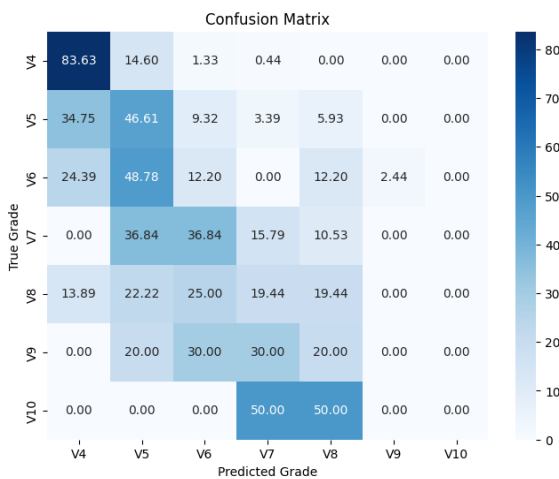
### 5.1.1 2016 Training Set

In this section, the evaluation takes place after solely training the model using data from the 2016 Moonboard set.

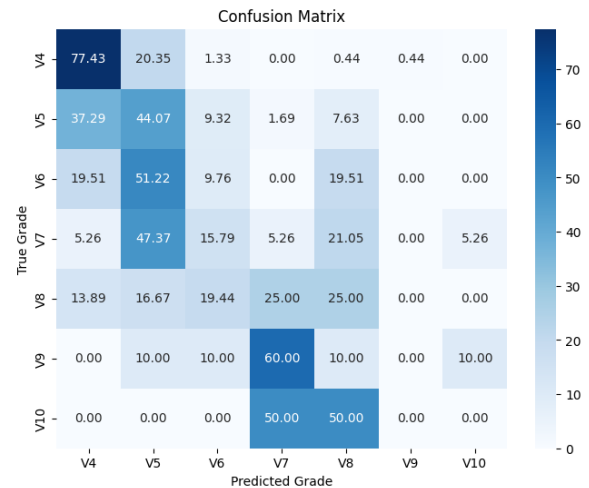
**Table 1:** The following documents the performance and evaluation of the model trained using data from the 2016 set. The test sets are derived from each of the Moonboard sets, with *HLP* denoting Human-Level Performance.

	HLP	2016 Test Set	2017 Test Set
<b>Accuracy</b>	45.0%	57.3%	53.3%
<b>Accuracy (+/- 1)</b>	87.5%	84.5%	84.3%

**Figure 12:** Confusion Matrix (2016)



**Figure 13:** Confusion Matrix (2017)



Perhaps unsurprisingly, the model performed best when evaluated on the same Moonboard set on which it was trained (**2016**), however, both sets still exceeded the *HLP* when assessed to the exact grade. Although, when evaluating to *plus or minus one* grade, both test sets fall short of the benchmark. Also, there's far less disparity between both test sets when given the one grade leeway.

Both confusion matrices display a similar trend of under-grading problems, with most incorrectly labelled problems falling below the diagonal intersection. Furthermore, the model performs similarly well on the lower grade problems, but struggles at the higher end of the spectrum, with no V9 or V10 problems being graded correctly.

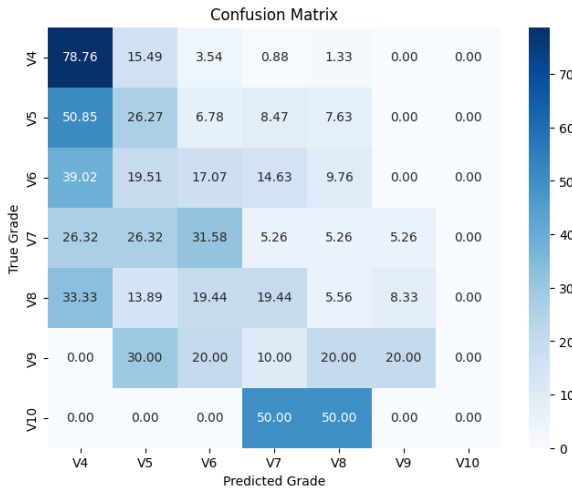
### 5.1.2 2017 Training Set

For this section, the model evaluated has been trained solely on problems from the Moonboard 2017 dataset.

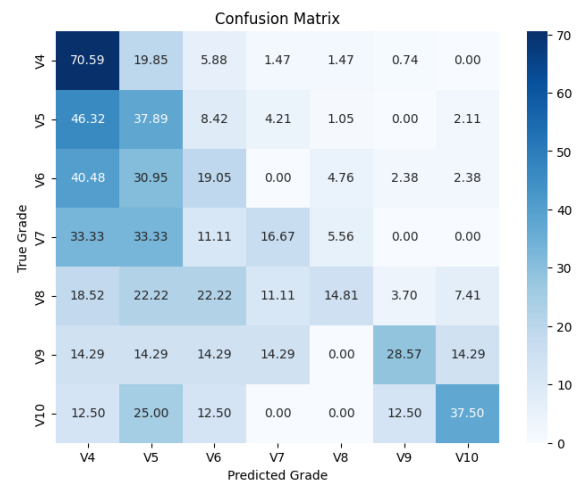
**Table 2:** The following documents the performance and evaluation of the model trained using data from the 2017 set. The test sets are derived from each of the Moonboard sets.

	HLP	2016 Test Set	2017 Test Set
<b>Accuracy</b>	45.0%	44.3%	46.0%
<b>Accuracy (+/- 1)</b>	87.5%	77.9%	78.4%

**Figure 14:** Confusion Matrix (2016)



**Figure 15:** Confusion Matrix (2017)



Similar to the previous evaluation, the model performed best on the set it was trained on (2017), though the difference in performance between the two sets was somewhat narrower. Interestingly, the models' performance on the 2017 set was inferior to the previous model trained on the 2016 set. However, unlike the previous assessment, only one of the two sets exceeded the *HLP* benchmark, this time with far less of a margin (1%). Once more, the model fell short of the *HLP* benchmark on both test sets when graded within a one-grade range, and this time the difference was even more pronounced.

Again observing the confusion matrices, both *test* sets displayed vast under-grading throughout the grade range. Although a noticeable diagonal (linear) trend is visible within the 2017 test set. One caveat of the model was the increased performance at the higher end of the grade spectrum, with far greater V9 and V10s being correctly labelled.

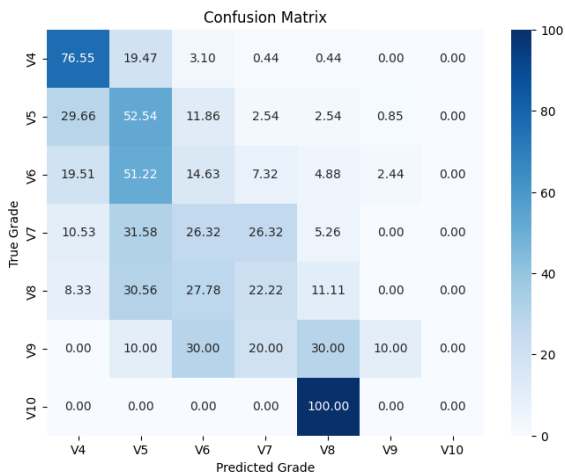
### 5.1.3 Mixed Training Set

In the following section, the model has been trained on a combination of both the 2016 and 2017 Moonboard sets.

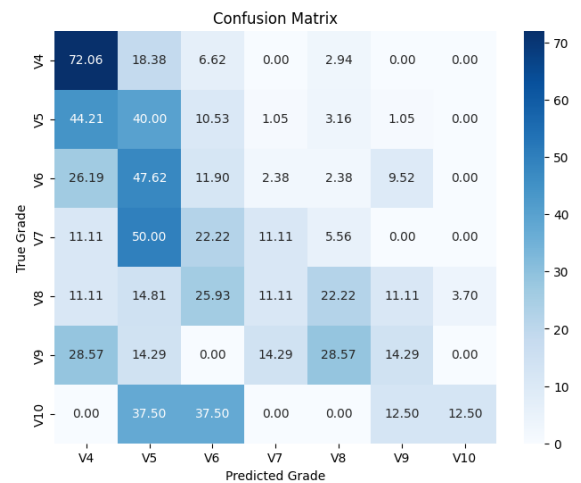
**Table 3:** The following documents the performance and evaluation of the model trained using data from the both 2017 and 2016 sets. The test sets are derived from each of the Moonboard sets.

	HLP	2016 Test Set	2017 Test Set
<b>Accuracy</b>	45.0%	51.1%	44.7%
<b>Accuracy (+/- 1)</b>	87.5%	83.4%	76.6%

**Figure 16:** Confusion Matrix (2016)



**Figure 17:** Confusion Matrix (2017)



With the model trained on an equal mix of both datasets, it is perhaps surprising to see the greatest disparity yet between the **2016** and **2017** test sets. A similar trend of the model outperforming the *HLP* on at least one of the test sets is continued, while again falling short on both for the *plus or minus one* grade accuracy.

The confusion matrices observe the usual under-grading, however, the spread on the **2016** set is much narrower than any of the previous assessments. Although the predictions are more chaotic in the **2017** set, the model does perform better at the higher end of the grade range.

## 5.2 Further Analysis

Throughout the evaluation process, a number of consistent observations were made. Without rigorous further testing, determining the cause of these may only be speculative, although following the principles of *Occam's Razor*\*, the most likely causes will be considered.

\* *Occam's Razor is a principle that suggests choosing the simplest explanation or solution when faced with multiple competing options, as it often tends to be the most accurate and efficient*

### 5.2.1 Under-Grading

Consistently, each model developed would under-grade the problems throughout the grade range. Why is this? One possibility is the *imbalance* within the datasets. Each dataset would contain far more problems at the lower end of the grade range, i.e. V4, V5 and V6. This skewness within the training set may have led to *bias* within the model. Since the model was exposed to a more extensive collection of lower-grade data during training, it may have developed a stronger understanding of these patterns, which inadvertently affects its grading of higher-grade problems. This is particularly apparent in the case of the **2016**-trained models and test sets, where none of the problems in the V9 to V10 range were correctly labelled. Why would the models trained upon and evaluated on the **2016** set struggle more? One plausible explanation lies in the reduced diversity of hold types present in this dataset. With fewer potential hold types available, a kind of compression effect occurs, where hold types alone no longer decisively determine a problem's difficulty. For instance, within the V7 to V10 range, climbers often encounter extraordinarily small holds. However, it's not the type of holds, but rather more intricate factors, such as body positioning, that truly define the difficulty - factors that may not be adequately captured by the model's inputs.

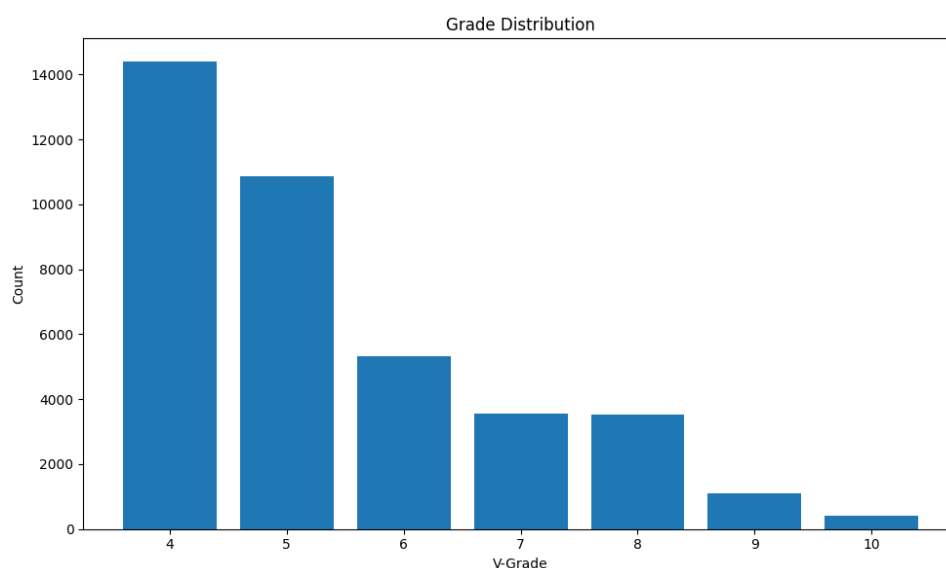


Figure 18: Grade Distribution



Imbalance within datasets can often be mitigated by techniques such as *oversampling* and *undersampling*. However, both these techniques were ruled out. *Undersampling* would be impractical as there are so few problems at the higher grade range; by applying undersampling uniformly across all grades, it would significantly shrink the dataset's size, greatly limiting the model's ability to learn from the available data. This would ultimately hinder the model's capacity to generalise effectively. *Oversampling* on the other hand would not have been suitable due to the *discreteness* of the data. Introducing noise and synthetic moves to the problems could completely change the difficulty and grade.

An attempt was made to enhance the model's ability to classify middle to higher-grade climbing problems through the use of a neural network *ensemble*. The initial network in this *ensemble* aimed to categorise problems as either *Easy* (V4-V5), *Medium* (V6-V8), or *Hard* (V9-V10). The intention was to create three more balanced datasets, each focused on a specific difficulty level. However, this approach again encountered challenges related to under-grading, as it often assigned no problems to the *Hard* category, misclassifying them as either *Medium* or *Easy* instead. Thus this approach was disregarded.

### 5.2.2 Training Set Impact

Each model performed noticeably better when tested on data from the same set as which the model was trained on. This is often referred to as *data dependency* or *domain specificity*. As with the under-grading, this issue may have occurred due to differences in the hold type distribution; with the **2016** set containing less variety of holds, a greater emphasis may have been placed on other features when determining the difficulty of a problem.

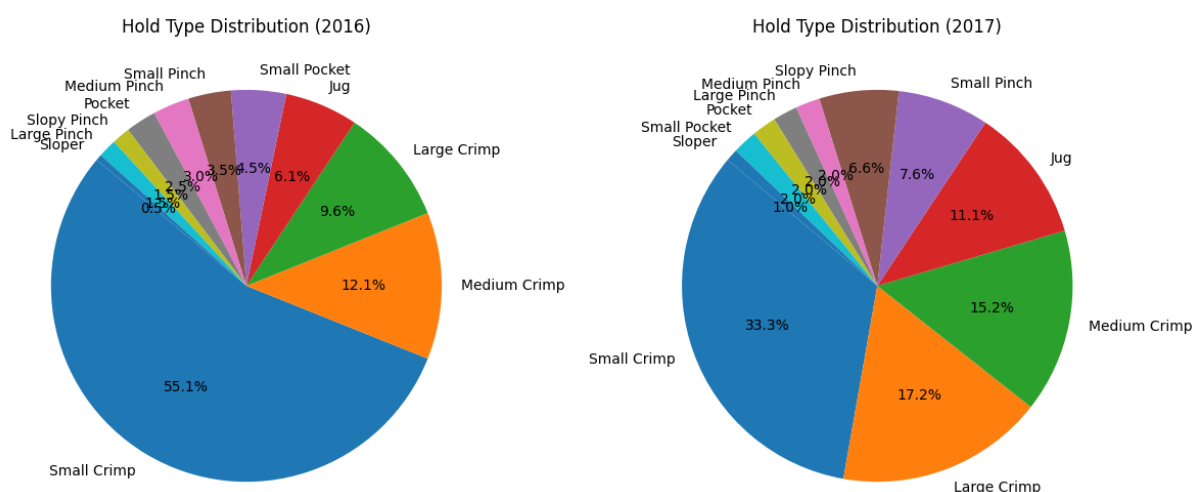


Figure 19: Hold Type Distribution

Under this assumption, it might be conjectured that the variety of holds in the **2017** act as a *red herring*\*, with too much emphasis being assigned to hold types. This disproportionate association between hold type and difficulty might explain the **2016** set outperforming the **2017** set, even when evaluated on the **2017** test set. Although, as stated, this is just speculation, and other potential causes such as the need for greater *hyperparameter* tuning may have been responsible.

\* A "red herring" is a misleading or distracting element introduced to divert attention from the main issue or argument.

### 5.3 Conclusion

As stated previously, when determining the success of the project, the initial objective should be referenced and evaluated. When undertaking this project, the aim was to train and produce a model capable of "performing well" on multiple Moonboard sets; an aspect which had not yet been explored within previous literature.

When evaluating the models, the *Human-Level Performance* was assigned as a benchmark for "performing well". With regards to this, a level of success can be concluded as the model trained on the **2016** dataset not only outperformed the benchmark on the set it was trained upon, but on the separate Moonboard set also. However, this metric of human performance was only derived from the person observing the problem, as opposed to predicting the grade after having the opportunity to climb it. Perhaps it would be facetious to conclude success without also considering the metric determined by *Dobles et al (2017)*, where human performance was measured at 93.4% accuracy. In light of this, achieving the objective of creating a model to aid in establishing the grade of a *first ascent* may be considered less successful.

However, success is not discrete. Concluding the project as successful or unsuccessful would be detrimental to progress. Expecting near or greater than human performance while conducting an initial exploration into transfer learning would be unrealistic. Instead the project can be considered as an altruistic effort to further understanding towards transfer learning within climbing and establishing a base point for further research to be conducted.

## 6 Limitations and Further Work

Throughout the project a number of limitations would be identified. Often resolving them would represent too great a challenge given the available time frame. However, these limitations may be the basis of future work and solving them could result in greater performance.

### 6.1 The Sequencer

The sequencer was fundamental to the development of the project. Without it, the features identified could not have been utilised. However, with the sequencer forming the basis of the work, any limitations surrounding it may have grave implications further down the pipeline. The sequencer was evaluated to have an 85% accuracy when predicting the sequence of a given climb, although the test set only consisted of 20 climbs, and therefore may not have been representative of the dataset as a whole. Assuming this accuracy held *true* for the entirety of the dataset, it would still imply that 15% of climbs would be sequenced incorrectly.

How could the performance of the sequencer be improved? A number of limitations were accepted during the development stage, but these could potentially be resolved within future work. One major limitation was the sequencers' assumption that all *problems* would start with the left hand holding the hold to the left and *vice versa*. This assumption, while often *true*, does not represent all *problems*. There exists a subset of problems where the climber is expected to start with their hands *crossed-over*, i.e. the right hand on the left hand side hold and *vice versa*. Another existing limitation is the punishment of *bumping*, i.e. moving the same hand twice in a row, within the algorithm. A negative weight was associated with moving the same hand twice as it would often result in non-natural movement and sequences. However, *bumping* is occasionally an essential move within climbing. It is considered more of an advanced movement, hence it's more likely to only exist within the higher grade ranges. The exclusion may be partly responsible for the struggles the model faced at predicting higher grade *problems*.

How could these limitations be mitigated? With future research, it may be necessary to carry out the tedious task of producing and labelling a dataset which contains correct sequences from which a Machine Learning algorithm could learn from. This would of course require the researcher to be an expert themselves in climbing, or rely on videos of others climbing each problem.

## 6.2 Hold Types

When labelling the hold types on each of the Moonboard sets, an almost arbitrary approach was taken. There were no set criterias for which the holds would be assigned to a certain category other than my own intuition. Not only this, but the categories themselves were entirely fabricated through my own judgement. This may have limited the overall success as certain holds may have been assigned to the wrong category. In particular the holds on the **2016** Moonboard set may have been misrepresented as I did not have the opportunity to view the set with my own eyes. Instead, the labels were produced from Youtube videos of others climbing on the board and my own experience of climbing on the **2017** set.

With future work, it may help to define set boundaries when determining hold types. The size (how far the hold extends from the wall) would be an obvious criteria to categorise holds, however, the lack of uniformity in hold shape and the difficulty of measuring holds while they are placed on the wall may present a challenge.

## 6.3 Future Work

Improving upon the limitations mentioned above would certainly be beneficial, but perhaps more interesting are the other routes that could be explored within future work. Within this subsection, a number of recommendations will be made for future researchers to consider.

### 6.3.1 to Recurrent Neural Networks and LSTM

Due to time constraints, only one machine learning approach was applied and evaluated. Given more time it would have been interesting to compare this approach with other methods, such as Convolutional Neural Nets.

### 6.3.2 Alternate Features

Minimal features were deliberately chosen to reduce the complexity of the problem, aiding in model generalisation. However, the success of future work could be further enhanced by exploring the inclusion of additional features that capture more nuanced aspects of the problem space, for example, hold orientation.

### 6.3.3 Real Rock and Other Standardised Boards

Expanding on the domain of Moonboards to other standardised boards, such as the popular *Kilter board* and *Tension board*, may also have been an interesting road to explore. With the generalised features selected, the pipeline of sequencer to training would be still entirely

relevant. Perhaps, of even greater interest is the application to the real world. Although the inherent limitation of real rock not being consistent in its angle would present a challenge.

## 7 Pragmatism & gradePredictor

Although pragmatism was not the primary concern of this research, a final deliverable was suggested. For this, *gradePredictor* would be developed. This application will allow users to enter the Moonboard set they wish to grade for and a set of holds. From this the grade will be predicted by applying the *2016Model* (as this achieved the greatest accuracy across both sets).

### 7.1 gradePredictor

For the user to navigate the various options presented, a *command line interface* (CLI) would be implemented. A CLI was chosen for its simplicity in both development and use. Three separate inputs would be required from the user, namely: The **year** (of the Moonboard set), the **starting holds** and the **subsequent holds**. These would be handled by a series of questions posed to the user. See below:

#### Step 1: Defining the year

```
Please enter Moonboard year (2016 or 2017): 2017
```

#### Step 2: Enter the start holds

```
Enter the start holds as a list of hold positions. Separate each position with a space.  
For example: 'F6 H5'  
Enter start holds: F6 H5
```

#### Step 3: Enter the subsequent holds

```
Enter the remaining holds as a list of hold positions. Separate each position with a space.  
For example: 'F8 J11 E13 C16 B18'  
Enter remaining holds: F8 J11 E13 C16 B18
```

After the inputs have been collected, the set of holds are passed on to a modified version of the sequencer. The year determines which hold set to use within this class. The modified sequencer (**sequencer2**) converts them to a sequence of *moves* from which the model can predict from. Finally, the predicted grade is outputted.

```
1/1 [=====] - 1s 837ms/step  
Predicted Grade: V4
```

## 7.2 Limitations

The simplicity of *gradePredictor* lends itself towards robustness. Validation has been entered at each level to prevent the user from entering invalid data. However, with the application being derived from the earlier work, many of the same limitations still hold true. One glaring issue is the potential future redundancy caused by the new Moonboard sets being developed. As of now, only the 2016 and 2017 sets are valid. The existing 2019 set would not be applicable as the hold types have not been labelled. Although, resolving this limitation would only require minor work to the back-end of the application.

Another potential limitation is the lack of usability. Although the CLI is inherently intuitive, many users prefer interacting with a more *graphical user interface* (GUI), especially those who are less *tech-literate* (J.W. Chen & J Zhang, 2007).

The app, in its current form as a Jupyter notebook (ipynb), lacks mobile support. While it's theoretically accessible on mobile devices, the practicality of using it in a climbing gym is limited. Climbers are unlikely to carry their PCs or laptops with them to the gym, which means they would need to wait until they're home to predict grades, often rendering it impractical

## 7.3 Conclusion

In evaluating the success of *gradePredictor*, it is essential to revisit the initial research objectives. The latter part of our *research question* aimed to assess the potential for *enhancing climbers' experiences* by ensuring grading *consistency*. In this regard, partial success can be claimed. *gradePredictor* empowers users to predict climb grades, offering a valuable second opinion that aids users in assessing a *problem's* true difficulty.

This partial success is attributed to the *2016Model's* improvement upon human level performance. However, it should be noted that the improvement was only apparent when climbers were not given the opportunity to attempt the *problem* beforehand. When given the chance to climb the *problems*, human-level performance far exceeded the model's capabilities. Although it should be considered that these human performance figures were determined from experienced climbers. Perhaps, a less experienced climber would not be capable of predicting to such accuracy, and for them the app would be of more use.

Ultimately, *gradePredictor's* success is intricately linked to the earlier development of the *2016Model*. While the model excels in problem analysis, its performance may vary depending on climbers' prior experiences and whether they have attempted the climb. This distinction underscores the potential value of *gradePredictor*, especially for climbers seeking a second opinion on a problem's grade.

## References

- Bourne, Roger, et al. "Measuring lifting forces in rock climbing: Effect of hold size and fingertip structure." *Journal of applied biomechanics* 27.1 (2011): 40-46.
- Brand, H. V. D. (2019). *Climbing Creativity: Teaching a Neural Network to Create Routes for the Moonboard Training Board*.
- Chen, J. W., & Zhang, J. (2007). Comparing text-based and graphic user interfaces for novice and expert users. In *AMIA annual symposium proceedings* (Vol. 2007, p. 125). American Medical Informatics Association.
- Creswell, J. W. (2014). *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage Publications.
- Dobles, A., Sarmiento, J. C., & Satterthwaite, P. (2017). Machine learning methods for climbing route classification. Web link: <http://cs229.stanford.edu/proj2017/finalreports/5232206>. Pdf.
- Duh, Y. S., & Chang, R. (2021). Recurrent neural network for moonboard climbing route classification and generation. *arXiv preprint arXiv:2102.01788*.
- Giles, L. V., Rhodes, E. C., & Taunton, J. E. (2006). The physiology of rock climbing. *Sports medicine*, 36, 529-545.
- Hochreiter, Sepp, and Jurgen Schmidhuber. "Long term short memory." *Neural Comput* 9.8 (1997): 1735-1780.
- Martelli, D., Tarchi, D., & Lippi, M. (2018). Automatic prediction of boulder problem difficulty in climbing gyms. In *Proceedings of the 26th ACM Conference on User Modeling, Adaptation and Personalization* (pp. 95-103).
- Medernach, J., Kleinöder, H., & Lötzerich, H. EFFECT OF MOONBOARD TRAINING ON GRIP STRENGTH IN BOULDERING.
- Pearce, L. A., Togher, J. A., & Mahaffey, R. G. (2020). Neural networks for sport climbing route grading. In *Proceedings of the 2020 ACM SIGKDD Workshop on Causal Discovery* (pp. 5-11).
- Quaine, F., & Martin, L. (1999). A biomechanical study of equilibrium in sport rock climbing. *Gait & Posture*, 10(3), 233-239.
- Seal, D., & Seal, R. (2022). Optimum Route Computation in a Chaotic Artificial Climbing Wall. In *ICT Systems and Sustainability: Proceedings of ICT4SD 2021, Volume 1* (pp. 671-680). Springer Singapore.

- Stapel, F. T. A. (2020). A heuristic approach to indoor rock climbing route generation (Bachelor's thesis, University of Twente).*
- Tai, C. H., Wu, A., & Hinojosa, R. (2020). Graph neural networks in classifying rock climbing difficulties. Student project report, CS, 230.*
- Vigouroux, L., Devise, M., Cartier, T., Aubert, C., & Berton, E. (2019). Performing pull-ups with small climbing holds influences grip and biomechanical arm action. Journal of sports sciences, 37(8), 886-894.*

## **Appendix A: Ethics Review (UREC1)**



## **UREC 1 RESEARCH ETHICS REVIEW FOR STUDENT RESEARCH WITH NO HUMAN PARTICIPANTS OR DIRECT COLLECTION OF HUMAN TISSUES, OR BODILY FLUIDS.**

All University research is required to undergo ethical scrutiny to comply with UK law. The University Research Ethics Policy ([www.shu.ac.uk/research/excellence/ethics-and-integrity/policies](http://www.shu.ac.uk/research/excellence/ethics-and-integrity/policies)) should be consulted before completing this form. The initial questions are there to check that completion of the UREC1 is appropriate for this study. The supervisor will approve the study, but it may also be reviewed by the College Teaching Program Research Ethics Committee (CTPREC) as part of the quality assurance process (additional guidance can be obtained from your College Research Ethics Chair<sup>1</sup>).

The final responsibility for ensuring that ethical research practices are followed rests with the supervisor for student research.

Note that students and staff are responsible for making suitable arrangements to ensure compliance with the General Data Protection Regulations (GDPR), for keeping data secure and if relevant, for keeping the identity of participants anonymous. They are also responsible for following SHU guidelines about data encryption and research data management. Guidance can be found on the SHU Ethics Website [www.shu.ac.uk/research/excellence/ethics-and-integrity](http://www.shu.ac.uk/research/excellence/ethics-and-integrity)

Please note that it is mandatory for all students to only store data on their allotted networked F drive space and not on individual hard drives or memory sticks etc.

This form also enables the University and College to keep a record confirming that research conducted has been subjected to ethical scrutiny. Students should retain a copy for inclusion in their research projects, and a copy should be uploaded to the relevant module Blackboard site.

The form must be completed by the student and approved by supervisor and/or module leader (as applicable). In all cases, it should be counter-signed by the supervisor and/or module leader and kept as a record showing that ethical scrutiny has occurred. Students should retain a copy for inclusion in the appendices of their research projects, and a copy should be uploaded to the module Blackboard site for checking.

Please note that it may be necessary to conduct a health and safety risk assessment for the proposed research. Further information can be obtained from the University's Health and Safety Website <https://sheffieldhallam.sharepoint.com/sites/3069/SitePages/Risk-Assessment.aspx>

---

<sup>1</sup> College of Social Sciences and Arts – Dr. Antonia Ypsilanti ([a.ypsilanti@shu.ac.uk](mailto:a.ypsilanti@shu.ac.uk))  
College of Business, Technology and Engineering – Dr. Tony Lynn ([t.lynn@shu.ac.uk](mailto:t.lynn@shu.ac.uk))  
College of Health, Wellbeing and Life Sciences – Dr. Nikki Jordan-Mahy ([n.jordan-mahy@shu.ac.uk](mailto:n.jordan-mahy@shu.ac.uk))  
)

## ARE YOU COMPLETING THE CORRECT FORM?

Does this study include collecting data or samples from human participants. YES/NO

Is the secondary data used in this study of a sensitive or contentious nature, or does it allow the identification of individuals or organisations (e.g., companies, school, councils, communities). YES/NO

If you have answered **YES** to either of these two questions you must complete a UREC2, 3 or 4 as appropriate.

### 1. General Details

<b>Details</b>	
Name of student	Thomas Hughes
SHU email address	b6027480@shu.ac.uk
Department/College	Computing
Name of supervisor	Chris Bates
Supervisor's email address	c.d.bates@shu.ac.uk
Title of proposed research	Transfer Learning and Moonboard Climbing
Proposed start date	15/06/23
Proposed end date	14/09/23
Brief outline of research to include, rationale (reasons) for undertaking the research & aims, and methods (max 500 words).	<p>One significant limitation in current research is its lack of foresight. The models produced are restricted to the specific Moonboard set on which they were trained, rendering them quickly outdated as new Moonboard sets are continually introduced</p> <p>This project aims to explore the application of transfer learning across multiple Moonboard sets by developing a predictive model capable of accurately grading climbing problems. This endeavour not only enhances the climber's experience but also lays the groundwork for future research, mitigating issues related to redundancy.</p>

## 2. Research in External Organisations

Question	Yes/No
1. Will the research involve working with/within an external organisation (e.g., school, business, charity, museum, government department, international agency, etc.)?	No
2. If you answered YES to question 1, do you have granted access to conduct the research? <i>If YES, students please show evidence to your supervisor. PI should retain safely.</i>	N/A
3. If you answered NO to question 2, is it because: A. you have not yet asked B. you have asked and not yet received an answer C. you have asked and been refused access. <i>Note: You will only be able to start the research when you have been granted access.</i>	N/A

## 3. Research with Products and Artefacts

Question	Yes/No
1. Will the research involve the use of specialist copyrighted documents, films, broadcasts, photographs, artworks, designs, products, programs, databases, networks, processes, existing datasets, or secure data?	No
2. If you answered YES to question 1, are the materials you intend to use in the public domain? <i>Notes: 'In the public domain' does not mean the same thing as 'publicly accessible'.</i> <ul style="list-style-type: none"> <li>Information which is 'in the public domain' is no longer protected by copyright (i.e., copyright has either expired or been waived) and can be used without permission.</li> <li>Information which is 'publicly accessible' (e.g., TV broadcasts, websites, artworks, newspapers) is available for anyone to consult/view. It is still protected by copyright even if there is no copyright notice. In UK law, copyright protection is automatic and does not require a copyright statement, although it is always good practice to provide one. It is necessary to check the terms and conditions of use to find out exactly how the material may be reused etc.</li> </ul> <i>If you answered YES to question 1, be aware that you may need to consider other ethics codes. For example, when conducting Internet research, consult the code of the Association of Internet Researchers; for educational research, consult the Code of Ethics of the British Educational Research Association.</i>	N/A

Question	Yes/No
3. If you answered NO to question 2, do you have explicit permission to use these materials as data? <i>If YES, please show evidence to your supervisor.</i>	N/A
4. If you answered NO to question 3, is it because: A. you have not yet asked permission B. you have asked and not yet received and answer C. you have asked and been refused access. <i>Note: You will only be able to start the research when you have been granted permission to use the specified material.</i>	N/A

**4. Does this research project require a health and safety risk assessment for the procedures to be used? (Discuss this with your supervisor)**

- ☐ Yes  
☒ No

If **YES** the completed Health and Safety Risk Assessment form should be attached. A standard risk assessment form can be generated through the Awaken system (<https://shu.awaken-be.com>). Alternatively if you require more specific risk assessment, e.g. a COSHH, attach that instead.



## Insurance Check

The University's standard insurance cover will not automatically cover research involving any of the following:

- i) Participants under 5 years old
- ii) Pregnant women
- iii) 5000 or more participants
- iv) Research being conducted in an overseas country
- v) Research involving aircraft and offshore oil rigs
- vi) Nuclear research
- vii) Any trials/medical research into Covid 19

If your proposals do involve any of the above, please contact the Insurance Manager directly ([fin-insurancequeries-mb@exchange.shu.ac.uk](mailto:fin-insurancequeries-mb@exchange.shu.ac.uk)) to discuss this element of your project.

## Adherence to SHU Policy and Procedures

<b>Ethics sign-off</b>	
<b>Personal statement</b>	
I can confirm that: <ul style="list-style-type: none"><li>• I have read the Sheffield Hallam University Research Ethics Policy and Procedures</li><li>• I agree to abide by its principles.</li></ul>	
<b>Student</b>	
Name: Thomas Hughes	Date: 14/06/2023
Signature: 	
<b>Supervisor ethical sign-off</b>	
I can confirm that completion of this form has confirmed that this research does not involve human participants. The research will not commence until any approvals required under Sections 2 & 3 have been received and any health and safety measures are in place.	
Name: Chris Bates	Date: 12/06/2023
Signature: 	
<b>Independent Reviewer ethical sign off</b> (if required to permit publication of findings with supervisor co-authorship).	
Name:	Date:
Signature:	

## Appendix B: Publication Form



College of Business,  
Technology and  
Engineering

Research Skills and  
Dissertation Module  
(55-706556).

### PUBLICATION PROCEDURE FORM

In this module, while you create your own research question or topic area, your supervisor makes a significant intellectual contribution to this work as the research progresses. Your supervisor will make the decision on whether your work merits publication based on the quality of the work you have produced. Your supervisor will co-author the paper for publication with you and your supervisor will both be listed as authors. You are required to sign the declaration below to confirm that you understand and will follow this procedure.

Declaration:

I, <u>Thomas Hughes</u> confirm that I understand will comply with the Publication Procedure outlined in the Module Handbook and the Blackboard Site.		
G5		
<b>Student:</b>	Signature <i>Tom Hughes</i>	Date 14/09/23
<b>Supervisor:</b>	Signature	Date

## Appendix C: Code

All code will be made available publicly on my Github repository:

<https://github.com/thughes1/Dissertation>

However, in the interest of transparency I will also paste the code here in the appendix.

### Sequencer:

```
import pandas as pd
import math
import ast
import random

##### Test Sets
#####
# POOOOOP
#sequence = [{'LH': 'G2'}, {'RH': 'K4'}]
#usableHolds = [{'I7': 7.0}, {'I8': 3.0}, {'A9': 6.5}, {'G11': 6.0}, {'F12': 4.0}, {'C14': 3.0}, {'E16': 7.5}, {'D18': 6.5}]
# REDLINE
sequence = [{'LH': 'F6'}, {'RH': 'H5'}]
usableHolds = [{'F8': 6.5}, {'J11': 3.0}, {'E13': 3.0}, {'C16': 3.0}, {'B18': 3.0}]
# MB MASTERS OPEN 2
#sequence = [{'LH': 'B4'}, {'RH': 'E6'}]
#usableHolds = [{'F8': 6.5}, {'H10': 3.0}, {'G13': 4.0}, {'E13': 3.0}, {'G16': 7.5}, {'J18': 7.5}]

#####
#####
def getHolds(moves):
    """ Returns the starting holds and usable holds """
    startHolds = []
    usableHolds = []
    for hold in moves:
        diff = getDiffVals(hold['Hold Type'])
        if hold['IsStart']:
            startHolds.append(hold['Position'])
        else:
            usableHolds.append({hold['Position']: diff})

    if len(startHolds) == 1:
        startHolds = [{'RH': startHolds[0]}, {'LH': startHolds[0]}]
    else:
        startHold1 = startHolds[0]
        startHold2 = startHolds[1]
```

```

# Checks which is left and which is right, assumes that start is not crossed.
if startHold1 > startHold2:
    startHolds = [{'RH':startHold1},{'LH':startHold2}]
else:
    startHolds = [{'LH':startHold1},{'RH':startHold2}]

return (startHolds, usableHolds)

def getDiffVals(holdType):
    """ Converts hold type to difficulty """
    difficulties = {
        "Small Crimp": 9.0,
        "Medium Crimp": 7.5,
        "Large Crimp": 6.5,
        "Pocket": 5.0,
        "Small Pocket": 9.0,
        "Jug": 3.0,
        "Small Pinch": 8.0,
        "Medium Pinch": 6.5,
        "Large Pinch": 4.0,
        "Slopy Pinch": 7.0,
        "Sloper": 7.5
    }
    return difficulties[holdType]

def getProblemNumber(problemName,df):
    """ Gets the index related to the problem name """
    return df.index[df['Name'] == problemName].tolist()

def getCoord(hold):
    """ Converts A1 to 1,1 etc """
    # Convert letter to column number
    x = ord(hold[0]) - ord('A') + 1
    # Convert number part to row number
    y = int(hold[1:])
    return (x, y)

def getDist(hold_A, hold_B):
    """ Calculates the distance between holds (Euclidean distance) """
    # This may need changing to centimeters if decide to go down the outdoor route
    holdA = getCoord(hold_A)
    holdB = getCoord(hold_B)
    squared_distances = [(p2 - p1) ** 2 for p1, p2 in zip(holdA, holdB)]
    sum_squared_distances = sum(squared_distances)
    distance = math.sqrt(sum_squared_distances)
    return distance

def getDiff(hold):

```



```

        return print(list(hold.values())[0])

def calcDiff(dist, diff):
    #distConst = 10
    #diffConst = 1
    #print(distConst)
    return (dist * distConst) + (diff * diffConst)

def setDistConst(val):
    global distConst
    distConst = val

def setDiffConst(val):
    global diffConst
    diffConst = val

def is_hold_to_left_or_right(hold_A, hold_B):
    """ Checks if hold_A is to the left or right of hold_B """
    coord_A = getCoord(hold_A)
    coord_B = getCoord(hold_B)

    # Compare x-coordinates of the holds
    if coord_A[0] < coord_B[0]:
        return "RH"
    elif coord_A[0] > coord_B[0]:
        return "LH"
    else:
        return "same"

def calcSequence(sequence, usableHolds, last_hand):
    startingUsableHolds = usableHolds[:]
    startingHolds = sequence[:]
    cross_over = 0 # Count cross-overs
    consecutive_cross_overs = 0 # Counter for consecutive cross-overs
    totalCrossOvers = 0
    maxSpan = 8 # Climbers max reach

    while True:
        min_diff = float('inf') # Initialize min_diff to positive infinity
        best_hold = None
        best_hand = None

        for hold in sequence:
            hand = list(hold.keys())[0]
            currHold = hold[hand]

            for usableHold in usableHolds:
                hold_loc = list(usableHold.keys())[0]

```

```

hold_diff = list(usableHold.values())[0]

distance = getDist(currHold, hold_loc)
hold_difficulty = calcDiff(distance, hold_diff)

# Check if the hold is intended as a foot hold (lower than the current hold)
if getCoord(hold_loc)[1] <= getCoord(currHold)[1]:
    hold_difficulty += 100

# Check if reaching hold exceeds the max span
currHands = [list(item.values())[0] for item in sequence[-2:]]
for position in currHands:
    if getDist(position, hold_loc) >= maxSpan:
        hold_difficulty += 100 # Make moving out of span impossible

if hand != last_hand: # Prioritize choosing a hold for the other hand
    if hold_difficulty < min_diff:
        min_diff = hold_difficulty
        best_hold = hold_loc
        best_hand = hand

if not best_hold:
    # If no holds are available for the other hand, choose any hold
    for usableHold in usableHolds:
        hold_loc = list(usableHold.keys())[0]
        hold_diff = list(usableHold.values())[0]

        # Check if the hold is intended as a foot hold (lower than the current hold)
        if getCoord(hold_loc)[1] < getCoord(sequence[-1][best_hand])[1]:
            continue

        if hold_diff < min_diff:
            min_diff = hold_diff
            best_hold = hold_loc
            best_hand = list(usableHold.keys())[0]

# Append the best hold to the sequence with the hand as the key
sequence.append({best_hand: best_hold})
usableHolds = [hold for hold in usableHolds if best_hold not in hold]

# Check for cross-overs
cross_over = 0 # Reset cross-over counter
for i in range(1, len(sequence)):
    prev_hand = list(sequence[i - 1].keys())[0]
    prev_hold = sequence[i - 1][prev_hand]

    curr_hand = list(sequence[i].keys())[0]
    curr_hold = sequence[i][curr_hand]

```

```

# Check for cross-over only if holds are intended for the other hand
if curr_hand != last_hand:
    #print('true')
    if is_hold_to_left_or_right(prev_hold, curr_hold) != curr_hand:
        cross_over += 1
        totalCrossOvers +=1
    else:
        #print('test')
        if cross_over != 0:
            cross_over -= 1

if cross_over >= 2:
    #print('true')
    consecutive_cross_overs += 1
    return calcSequence(startingHolds,startingUsableHolds,'RH')

last_hand = best_hand # Update the last hand used

if '18' in best_hold:
    break # Break the loop if '18' is found and added to the sequence
"""

print("Final sequence:")
print(sequence)
print("Number of cross-overs:", cross_over)
print("Consecutive cross-overs:", consecutive_cross_overs)
print("Total Crossovers:", totalCrossOvers)
"""

return sequence

def getHoldTypes(moves):
    """ Returns the hold types """
    holdTypes = []
    for move in moves:
        holdTypes.append({move['Position']: move['Hold Type']})
    return holdTypes

def convertSeq(sequence,holdTypes):
    """ Converts sequence to
    {(hold1,hold2,distance,holdType),...,(hold1,hold2,distance,holdType)} """
    moves = []
    # Split into individual moves, i.e. the two start holds and the target hold
    for i in range(len(sequence) - 2):
        hold_A = list(sequence[i].values())[0]
        hold_B = list(sequence[i+1].values())[0]
        hold_C = list(sequence[i+2].values())[0]
        # This is assuming the hand doesn't bump, which is very rare in the current model
        dist = getDist(hold_A,hold_C)

```

```

# Convert to hold types
holdType_A = next((item[hold_A] for item in holdTypes if hold_A in item), None)
holdType_B = next((item[hold_B] for item in holdTypes if hold_B in item), None)
holdType_C = next((item[hold_C] for item in holdTypes if hold_C in item), None)

move = (holdType_A, holdType_B, holdType_C, dist)
moves.append(move)
return moves

def main():
    df = pd.read_csv('Moonboard2016Problems.csv')
    # To acquire only the 2016 set
    df = df[df['MoonBoardHoldSetup'] == 'MoonBoard 2016']
    global usableHolds # Use the global variable
    global sequence
    last_hand = 'LH' # Initialize the last hand used to 'RH'
    # Initialise weights as equal
    setDiffConst(1)
    setDistConst(1)
    #problemName = 'WALK LIKE AN EGYPTIAN'
    setDiffConst(1)
    setDistConst(10)
    inputs = []
    for i in range(len(df)):
        print('%d/%d' % (i, len(df)), end='\r')
        try:
            problemName = df.loc[i, 'Name'] # Get the problem name from the 'Name' column
            problemNumber = getProblemNumber(problemName, df)
            moves = df.loc[problemNumber[0], 'Moves']
            holdTypes = getHoldTypes(ast.literal_eval(moves))
            holds = getHolds(ast.literal_eval(moves))
            startHolds = holds[0]
            usableHolds = holds[1]
            sequence = calcSequence(startHolds, usableHolds, last_hand)
            inputs.append(convertSeq(sequence, holdTypes))
        except:
            inputs.append(None)
        continue

    df['Input'] = inputs # Add the 'input' column to the DataFrame
    df.to_csv('problemSet2016.csv', index=False)

main()

```

## Model:

```
import numpy as np
import ast
import re
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Masking
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import precision_score
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dropout
from tensorflow.keras.regularizers import l2
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
```

```
def extract_repeats(text):
    """ Extracts number of repeats """
    numeric_value = re.search(r'\d+', text)
    if numeric_value:
        return int(numeric_value.group())
    else:
        return None
```

```
def remLowRepeats(df):
    """ Removes climbs with less than 20 repeats """
    # Apply the function to the 'RepeatText' column
    df['RepeatText'] = df['RepeatText'].apply(extract_repeats)
    return df[df['RepeatText']>=20]
```

```
def preprocess(df):
    # Define global test set (Unseen data) to ensure no leakage
    global testSet
    # 2017 set
    #df = pd.read_csv('trainingFinal.csv')
    # 2016 set
    #df = pd.read_csv('testSetFinal.csv')
    df = remLowRepeats(df)
```

```
    hold_types = ["Small Crimp", "Medium Crimp", "Large Crimp", "Pocket", "Small  
Pocket", "Jug", "Small Pinch", "Medium Pinch", "Large Pinch", "Slopy Pinch", "Sloper"]
```

```
    # Process input data for each problem in the DataFrame
```

```

encoded_inputs = []
grades = []

for input_data, grade in zip(df['Input'], pd.to_numeric(df['V_Grade'], errors='coerce')):
    try:
        input_data_list = ast.literal_eval(input_data)
        encoded_seq = []

        for seq in input_data_list:
            current_hold1, current_hold2, target_hold, distance = seq
            encoded_hold = [0] * (len(hold_types) + 1) # +1 for the distance

            if current_hold1 in hold_types:
                encoded_hold[hold_types.index(current_hold1)] = 1
            if current_hold2 in hold_types:
                encoded_hold[hold_types.index(current_hold2)] = 1
            if target_hold in hold_types:
                encoded_hold[hold_types.index(target_hold)] = 1

            # Append the distance as the last element of the encoded sequence
            encoded_hold[-1] = distance

            encoded_seq.append(encoded_hold)

        if not np.isnan(grade) and 4 <= int(grade) <= 10: # Only append if grade is not NaN
            and only take V4-V10
            encoded_inputs.append(encoded_seq)
            grade = int(grade)
            grades.append(grade)

    except:
        continue
    # Convert grades to a numpy array
    grades = np.array(grades)
    # Pad sequences with masking to a length of 12
    max_seq_length = 12
    padded_inputs = pad_sequences(encoded_inputs, padding='post', dtype='float32',
maxlen=max_seq_length)

    return padded_inputs, grades

def train():
    #padded_inputs, grades = preprocess(pd.read_csv('testSetFinal.csv'))
    padded_inputs, grades = preprocess(pd.read_csv('mixedTrain.csv'))
    # Convert grades to categorical labels
    num_classes = 7 # Number of grade categories (V4-V10)
    grades_categorical = np.zeros((grades.shape[0], num_classes))

```

```

for i, grade in enumerate(grades):
    grade_category = int(grade) - 4 # Assuming your grades are 3, 4, 5, ...
    grades_categorical[i, grade_category] = 1

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    padded_inputs, grades_categorical, test_size=0.2, random_state=None
)

# Create an LSTM model
model = Sequential()

# Add masking layer
model.add(Masking(mask_value=0.0, input_shape=(padded_inputs.shape[1],
padded_inputs.shape[2])))

# Generally less nodes works well for generalised problems
model.add(LSTM(128, return_sequences=False))

# Add Dense output layer with num_classes units and softmax activation
model.add(Dense(num_classes, activation='softmax'))

# Compile the model with categorical cross-entropy loss
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Create a ModelCheckpoint callback to save the model after the last epoch
checkpoint_callback = ModelCheckpoint(
    'mixedModel.h5',      # Filepath to save the model
    save_best_only=False, # Save the model after each epoch
    save_weights_only=False, # Save the entire model (including architecture)
    save_freq='epoch',    # Save after each epoch
    verbose=1             # Print messages about saving
)

# Train the model with the checkpoint callback and validation data (to check for
overfitting)
model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=15, batch_size=1,
callbacks=[checkpoint_callback])

return model

def evaluate(model, X_test, y_test):
    grade_labels = ["V4", "V5", "V6", "V7", "V8", "V9", "V10"]

    # Convert y_test to categorical labels
    num_classes = 7 # Number of grade categories (V4-V10)

```

```

y_test_categorical = np.zeros((y_test.shape[0], num_classes))
for i, grade in enumerate(y_test):
    grade_category = int(grade) - 4
    y_test_categorical[i, grade_category] = 1

# Make predictions using the model
y_pred = model.predict(X_test)
# Convert predicted probabilities to class labels
y_pred_class = np.argmax(y_pred, axis=1)

# Convert y_test_categorical back to original grade labels
y_test_original = np.argmax(y_test_categorical, axis=1)

# Calculate custom accuracy within +/- 1 grade range
correct_count = 0
correct_count_within_one = 0 # Initialize a count for within +/- one grade
for true_grade, pred_grade in zip(y_test_original, y_pred_class):
    if abs(true_grade - pred_grade) <= 1:
        correct_count_within_one += 1
    if true_grade == pred_grade:
        correct_count += 1
custom_accuracy = correct_count / len(y_test_original)
custom_accuracy_within_one = correct_count_within_one / len(y_test_original)

# Evaluate the model on the test set
test_loss = model.evaluate(X_test, y_test_categorical, verbose=0)

# Create a confusion matrix
confusion_mat = confusion_matrix(y_test_original, y_pred_class)

return custom_accuracy, custom_accuracy_within_one, test_loss[0], confusion_mat

def main():
    # Test Set
    padded_inputs, grades = preprocess(pd.read_csv('2017Test.csv'))
    padded_inputs = padded_inputs[:, :12, :] # Set the size to 12 s.t. consistent with
each dataset
    # Train model
    model = train()
    #evaluate(model, padded_inputs, grades)
    accuracy,plusminusone,_,confusion_mat = evaluate(model, padded_inputs, grades)

# Print the metrics
print(f"Custom Accuracy: {accuracy:.2%}")
print(f"Custom Accuracy Within +/- 1 Grade Range: {plusminusone:.2%}")

grade_labels = ["V4", "V5", "V6", "V7", "V8", "V9", "V10"]

```



```
# Normalise the confusion matrix to percentages
row_sums = confusion_mat.sum(axis=1, keepdims=True)
confusion_mat_percent = (confusion_mat / row_sums) * 100

# Visualise the confusion matrix using Seaborn's heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat_percent, annot=True, fmt=".2f", cmap="Blues",
            xticklabels=grade_labels, yticklabels=grade_labels)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Grade")
plt.ylabel("True Grade")
plt.show()
```

```
main()
```

## Sequencer2:

```
import pandas as pd
import math
import ast

def loadHoldTypesFromCSV(csv_filename):
    """ Load hold types from a CSV file into a dictionary """
    hold_types_dict = {}
    try:
        hold_types_df = pd.read_csv(csv_filename)
        for index, row in hold_types_df.iterrows():
            position = row['Hold']
            hold_type = row['Type']
            hold_types_dict[position] = hold_type
    except Exception as e:
        print(f"Error loading hold types: {str(e)}")
    return hold_types_dict

def getHolds(holds):
    """ Returns the starting holds and usable holds """
    startHolds = []
    usableHolds = []

    for hold in holds[1]: # Iterate through the "rest of the holds" list
        hold_type = hold.get('Hold Type', None)
        if hold_type:
            diff = getDiffVals(hold_type)
            position = hold.get('Position', None)
            if position:
                usableHolds.append({position: diff})

    for hold in holds[0]: # Iterate through the "start holds" list
        position = hold.get('Position', None)
        if position:
            startHolds.append(position)

    if len(startHolds) == 1:
        startHolds = [{ 'RH': startHolds[0] }, { 'LH': startHolds[0] }]
    else:
        startHold1 = startHolds[0]
        startHold2 = startHolds[1]
        # Checks which is left and which is right, assumes that start is not crossed.
        if startHold1 > startHold2:
```

```

startHolds = [{'RH': startHold1}, {'LH': startHold2}]
else:
startHolds = [{'LH': startHold1}, {'RH': startHold2}]

return (startHolds, usableHolds)

def getDiffVals(holdType):
    """ Converts hold type to difficulty """
    difficulties = {
        "Small Crimp": 9.0,
        "Medium Crimp": 7.5,
        "Large Crimp": 6.5,
        "Pocket": 5.0,
        "Small Pocket": 9.0,
        "Jug": 3.0,
        "Small Pinch": 8.0,
        "Medium Pinch": 6.5,
        "Large Pinch": 4.0,
        "Slopy Pinch": 7.0,
        "Sloper": 7.5
    }
    return difficulties[holdType]

def getCoord(hold):
    """ Converts A1 to 1,1 etc """
    # Convert letter to column number
    x = ord(hold[0]) - ord('A') + 1
    # Convert number part to row number
    y = int(hold[1:])
    return (x, y)

def getDist(hold_A, hold_B):
    """ Calculates the distance between holds (Euclidean distance) """
    # This may need changing to centimeters if decide to go down the outdoor route
    holdA = getCoord(hold_A)
    holdB = getCoord(hold_B)
    squared_distances = [(p2 - p1) ** 2 for p1, p2 in zip(holdA, holdB)]
    sum_squared_distances = sum(squared_distances)
    distance = math.sqrt(sum_squared_distances)
    return distance

def calcDiff(dist, diff):
    distConst = 10
    diffConst = 1

```

```

    return (dist * distConst) + (diff * diffConst)

def is_hold_to_left_or_right(hold_A, hold_B):
    """ Checks if hold_A is to the left or right of hold_B """
    coord_A = getCoord(hold_A)
    coord_B = getCoord(hold_B)

    # Compare x-coordinates of the holds
    if coord_A[0] < coord_B[0]:
        return "RH"
    elif coord_A[0] > coord_B[0]:
        return "LH"
    else:
        return "same"

def calcSequence(sequence, usableHolds, last_hand):
    startingUsableHolds = usableHolds[:]
    startingHolds = sequence[:]
    cross_over = 0 # Count cross-overs
    consecutive_cross_overs = 0 # Counter for consecutive cross-overs
    totalCrossOvers = 0
    maxSpan = 8 # Climbers max reach

    while True:
        min_diff = float('inf') # Initialize min_diff to positive infinity
        best_hold = None
        best_hand = None

        for hold in sequence:
            hand = list(hold.keys())[0]
            currHold = hold[hand]

            for usableHold in usableHolds:
                hold_loc = list(usableHold.keys())[0]
                hold_diff = list(usableHold.values())[0]

                distance = getDist(currHold, hold_loc)
                hold_difficulty = calcDiff(distance, hold_diff)

                # Check if the hold is intended as a foot hold (lower than the current hold)
                if getCoord(hold_loc)[1] <= getCoord(currHold)[1]:
                    hold_difficulty += 100

                # Check if reaching hold exceeds the max span

```

```

currHands = [list(item.values())[0] for item in sequence[-2:]]
for position in currHands:
    if getDist(position, hold_loc) >= maxSpan:
        hold_difficulty += 100 # Make moving out of span impossible

    if hand != last_hand: # Prioritize choosing a hold for the other hand
        if hold_difficulty < min_diff:
            min_diff = hold_difficulty
            best_hold = hold_loc
            best_hand = hand

if not best_hold:
    # If no holds are available for the other hand, choose any hold
    for usableHold in usableHolds:
        hold_loc = list(usableHold.keys())[0]
        hold_diff = list(usableHold.values())[0]

        # Check if the hold is intended as a foot hold (lower than the current hold)
        if getCoord(hold_loc)[1] < getCoord(sequence[-1][best_hand])[1]:
            continue

        if hold_diff < min_diff:
            min_diff = hold_diff
            best_hold = hold_loc
            best_hand = list(usableHold.keys())[0]

# Append the best hold to the sequence with the hand as the key
sequence.append({best_hand: best_hold})
usableHolds = [hold for hold in usableHolds if best_hold not in hold]

# Check for cross-overs
cross_over = 0 # Reset cross-over counter
for i in range(1, len(sequence)):
    prev_hand = list(sequence[i - 1].keys())[0]
    prev_hold = sequence[i - 1][prev_hand]

    curr_hand = list(sequence[i].keys())[0]
    curr_hold = sequence[i][curr_hand]

# Check for cross-over only if holds are intended for the other hand
if curr_hand != last_hand:
    if is_hold_to_left_or_right(prev_hold, curr_hold) != curr_hand:
        cross_over += 1
    totalCrossOvers += 1

```

```

        else:
            if cross_over != 0:
                cross_over -= 1

    if cross_over >= 2:
        consecutive_cross_overs += 1
    return calcSequence(startingHolds, startingUsableHolds, 'RH')

last_hand = best_hand # Update the last hand used

if '18' in best_hold:
    break # Break the loop if '18' is found and added to the sequence

return sequence

def convertSeq(sequence, holdTypes):
    """ Converts sequence to {(hold1, hold2, distance, holdType), ..., (hold1, hold2,
distance, holdType)} """
    moves = []
    for i in range(len(sequence) - 2):
        hold_A = list(sequence[i].values())[0]
        hold_B = list(sequence[i + 1].values())[0]
        hold_C = list(sequence[i + 2].values())[0]
        dist = getDist(hold_A, hold_C)

        # Look up hold types using hold positions
        holdType_A = holdTypes.get(hold_A, 'Unknown')
        holdType_B = holdTypes.get(hold_B, 'Unknown')
        holdType_C = holdTypes.get(hold_C, 'Unknown')

        move = (holdType_A, holdType_B, holdType_C, dist)
        moves.append(move)
    return moves

def main(holds, year):
    if year == '2016':
        holdTypes = loadHoldTypesFromCSV('holdType2016.csv')
    else:
        holdTypes = loadHoldTypesFromCSV('holdType2017.csv')
    """
    holds = [{ 'Position': 'F6' },
              { 'Position': 'H5' },
              { 'Position': 'F8' },
              { 'Position': 'J11' },

```

```

        {'Position':'E13'},
        {'Position':'C16'},
        {'Position':'B18'}]]
'''
    for hold in holds[0]:
        position = hold['Position']
        hold_type = holdTypes.get(position, 'Unknown') # Get the hold type or 'Unknown' if
not found
        hold['Hold Type'] = hold_type # Assign the hold type to the hold

    for hold in holds[1]:
        position = hold['Position']
        hold_type = holdTypes.get(position, 'Unknown') # Get the hold type or 'Unknown' if
not found
        hold['Hold Type'] = hold_type # Assign the hold type to the hold

    # Split the holds into startHolds and usableHolds
    startHolds, usableHolds = getHolds(holds)

    # Initialize the last_hand
    last_hand = 'LH'

    # Calculate the sequence
    sequence = calcSequence(startHolds, usableHolds, last_hand)

    moves = convertSeq(sequence,holdTypes)
    return moves

if __name__ == "__main__":
    main()

```

## **gradePredictor:**

```
import sequencer2
import pandas as pd
import re
import ast
import numpy as np
import pandas as pd
import warnings
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences

def convertToVGrade(number):
    """ Converts 3.0, 4.0, 5.0, etc., to V3, V4, V5, etc. """
    with warnings.catch_warnings():
        warnings.filterwarnings("ignore", category=FutureWarning)
    # Convert to int
    number = int(number)
    return 'V' + str(number)

def preprocess_input(input_data):
    hold_types = ["Small Crimp", "Medium Crimp", "Large Crimp", "Pocket", "Small
    Pocket", "Jug", "Small Pinch", "Medium Pinch", "Large Pinch", "Slopy Pinch", "Sloper"]

    encoded_seq = []

    for seq in input_data:
        current_hold1, current_hold2, target_hold, distance = seq
        encoded_hold = [0] * (len(hold_types) + 1) # +1 for the distance

        if current_hold1 in hold_types:
            encoded_hold[hold_types.index(current_hold1)] = 1
        if current_hold2 in hold_types:
            encoded_hold[hold_types.index(current_hold2)] = 1
        if target_hold in hold_types:
            encoded_hold[hold_types.index(target_hold)] = 1

        # Append the distance as the last element of the encoded sequence
        encoded_hold[-1] = distance

        encoded_seq.append(encoded_hold)

    # Pad sequences with masking to a length of 12
    max_seq_length = 12
    padded_input = pad_sequences([encoded_seq], padding='post', dtype='float32',
    maxlen=max_seq_length)
    return padded_input
```



```

def predict(moves):
    model = load_model("2016Model.h5")
    padded_input = preprocess_input(moves)
    if padded_input is not None:
        # Predict using the loaded model
        predicted_grade = model.predict(padded_input)
        predicted_grade_index = np.argmax(predicted_grade)
        predicted_grade_label = convertToVGrade(predicted_grade_index + 3) # Adjust for
your grade labels
        return predicted_grade_label
    else:
        return "Invalid input data"

def main():
    # Prompt the user to enter the year
    while True:
        # Prompt the user to enter the year
        year = input("Please enter Moonboard year (2016 or 2017): ")
        if year == '2016' or year == '2017':
            break # Exit the loop if a valid year is entered
        else:
            print("Please ensure the year entered is either 2016 or 2017")

    # Prompt the user to enter start holds
    print("Enter the start holds as a list of hold positions. Separate each position with a
space.")
    print("For example: 'F6 H5'")
    start_holds_input = input("Enter start holds: ").upper().split()

    # Create the start_holds list from user input
    start_holds = [{'Position': position} for position in start_holds_input]

    # Prompt the user to enter the rest of the holds
    print("Enter the remaining holds as a list of hold positions. Separate each position
with a space.")
    print("For example: 'F8 J11 E13 C16 B18'")
    remaining_holds_input = input("Enter remaining holds: ").upper().split()

    # Create the remaining_holds list from user input
    remaining_holds = [{'Position': position} for position in remaining_holds_input]

    # Combine start holds and remaining holds into holds_data
    holds_data = [start_holds, remaining_holds]

    # Call the function from sequencer2 module
    result = sequencer2.main(holds_data, year)

    # Call the predict function to predict the grade

```

```
grade = predict(result)
print("Predicted Grade:", grade)

if __name__ == "__main__":
    main()
```