Indoor Rock-Climbing Route Classification with Machine Learning

UNIVERSITY OF
LINCOLN

Thomas Moxom

MOX18674258

18674258@students.lincoln.ac.uk

A report submitted in partial fulfilment of the requirement for BSc(Hon) Computer Science

Supervisor: Dr Mubeen Ghafoor
Word Count: 12,011 (17171 including everything)

25th May 2022

Demonstration Link: https://youtu.be/ctsdhZa-8VQ

Acknowledgements

I would like to acknowledge my family and friends for their hindsight and positive assurance. Additionally, I would like to thank my supervisor Mubeen for his insight and resources that were integral to the outcome of this project.

Abstract

This project presents an artefact to classify indoor climbing routes from the MoonBoard into three grade boundaries. It presents the problem space of climbing route grading and the justification for this research. The model to classify is determined by comparing a range of pre-trained models with a developed convolutional neural network. Any future practitioners should be aware of the similarities between classes in the dataset which lead to confusion. The model is improved through a testing an optimisation strategy to achieve a score of 63% accuracy. The results of this project show promise for future development of a complete application for automatically grading MoonBoard routes and assist route-setting.

Keywords: Climbing Route/Problem, Route-Setting, Grade, Classification, and Machine Learning (ML).

Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

1.1 Project Background and Rationale

Climbing route grading is a difficult task and is always subject to debate, (Draper, 2016). This study attempts to resolve this problem by classifying routes with machine learning (ML). Using the MoonBoard, which is a standardised training tool used in climbing gyms; where routes selected from an application can be displayed using LEDs (Moon Climbing, 2019). This project samples the routes from the application and classifies them into difficulty groups. See Figure 2 for an example route from the MoonBoard.

The justification for this project stems from the inconsistency in route-setting from gym to gym (De Bruijn, 2019). E.g., the same grade can be very different across centres. Route setters judge what a route would be like at each level, so their experience weighs into the grade they assign (*ibid)*. This inconsistency detracts from the meaning of the grade attached and has led some centres to ignore the grade system entirely. If a standardised system for route-setting was developed, the grade of a route would be universally recognised as the definite grade it is given. Climbers can track their progress and not be disheartened when they fail a grade they can usually do.

Route-setting is a trial-and-error process that can take hours to get it right or at least close to the desired grade and vision. Due to this route-setters get tired and injured creating a route (Russell, 2021), that is achievable but also stretches each skill level of climber. Therefore, this project will help keep route-setters safe but also achieve their vision for a route. This project will make route setting not just easier and quicker for centres but assist people with home walls without professional training. Any climber with the wall, holds and tools can create a route at a grade of their choice.

1.2 Project Aims and Objectives

This paper attempts to answer the following research question by splitting tasks into an aim and the objectives to complete it. These have been expanded and updated throughout the development of this project. Additionally, the objectives are covered in the Gantt chart in Figure 1.

RQ1: Can indoor climbing routes be classified with ML?

Aim: to classify climbing routes by difficulty.
Objectives: to complete this aim:

- Research related projects on climbing problem classification for grades.
- Gather and pre-process a dataset, split into training & test data.
- Apply multiple methods to classify data into easy, medium, and hard routes.
- Test classification with new route image.
- Optimise classification model.
- Conclude and analyse results.

Stretch Objectives:

RQ2: Can indoor climbing routes be generated with ML?

Aim: to generate climbing of a desired difficulty.
Objectives: to complete this aim:

- Research related projects on climbing problem generation.
- Use this project model output into a new generative model.
- Attempt methods to generate new MoonBoard routes at different classes.
- Compare generated routes to actual routes with user testing.
- Optimise generative model.

RQ3: Can a portable application be made to generate and classify MoonBoard routes?

Aim: to build an application for users to create, generate, and grade MoonBoard routes.
Objectives: to complete this aim:

- Research related projects of applications with climbing functionality.
- Combine classification model and generative model into an application for mobile
- Test and case study with users and gather feedback.
- Optimise application.
- Package and release artefact.

1.3 Report overview

This document will form a technical report that summarises, justifies, evaluates, and contextualises the work undertaken for this project. It will demonstrate an implementation of an artefact developed; to attempt resolving the research questions and problems raised previously. It includes a critical evaluation and reflection of both the development process and learning requirements of completing this project.

Chapter 2

2 Literature Review

Phillips et al, (2011) lay the foundation for this field. They developed an assistance system for creating climbing routes henceforth, known as route setting. For when creativity blocks arise or train novice setters. Presenting a formalized Climbing Route Description Language (CRDL) and, a variation generator named Strange Beta. The results revealed the application can help produce routes at an equal or sometimes better quality than those traditionally set. The CRDL system could lead to many other interesting projects but is limiting, because the transcription does not cover footholds or other common movements. Kempen, (2018) applies this system later. To increase variation in the generator, the algorithm must use multiple input routes because a single route is too short for interesting enough variation.

Similarly, Pfiel et al, (2011) create an interactive route designer using a simulated virtual climber, to visualize route setting and grading for novice climbers and setters. The routes are separated into sequences of poses where the grade is calculated using heuristics and the movement between is interpolated. Results showed the system was useful for route setters creating child problems and avoiding the need for test climbing, as is custom for classic route design. The issue with this project is the difficulty is determined by hold spacing; not the move or hold types involved. However, the prospect of this project is intriguing for a quick and intuitive route designing system; usable for any experienced level climber/setter to make fun and interesting routes.

Next, Kosmalla et al, (2015) use wrist-worn Inertia Measurement Units (IMUs) to automatically recognise climbing routes. Used as a tool for training diaries, virtual coaches, and usage analytics for gyms – much like how activity trackers are used in other sports. Their results are promising but need refining. Route recognition relies on training data from other climbed routes, thus for effective recognition every route would have to be climbed beforehand to display useful information for new climbers. Currently the tracking does not distinguish between transitions from one hold to another – they propose these movements could be detected with ML methods such as climb segmentation shown by Ladha et al, (2013). These activity trackers are reused in similarly later by Ebert et al, (2017).

This research field has been explored before by Dobles et al, (2017). Building on the work of Phillips et al, (2011), they apply three ML methods to classify the grade of climbing routes from a MoonBoard dataset. Their implementation was extensive and demonstrated clear results of the classifiers compared to human classification. The downfalls from this project stem from the dataset. The MoonBoard consists of routes created by any user to the application therefore, the grade attached, and quality of climb is inconsistent. Additionally, the grade distribution in their data is uneven: 31% of routes are the easiest grade, and only 1% are in the hardest range. The best performing classifier CNN reaches an accuracy of 36.5%, to improve this they mention including a rating of each hold into the model. However, this relates to the promising progress shown by Krizhevsky et al, (2012) for image classification with CNNs.

Ebert et al, (2017) apply similar technology to Kosmalla et al, (2015) to measure movement of climbers to automatically assess and classify the grade of a climbing route. "The core skills *control, stability, speed and economical use of strength* are harder to achieve for difficult routes" Ebert et al, (2017, pg. 3) by sensing these measures they can assess the difficulty of a route with 98% success rate. It is worth noting that the study features low experience climbers on low difficulty routes, characteristics of hard routes not depicted at all within this feature set. Additionally, the study is measuring climbers on routes where the grades are set within overlapping boundaries thus, it is not known the true grade of the routes tested – which is a common theme in climbing route classification problems.

By utilising the CRDL and Strange Beta routes from Phillips et al, (2011), a second study was done by Kempen, (2018) using the system to predict the difficulty of a climbing route as either *Easy* or *Hard*. With six predictor variations, the best one achieved a disappointing accuracy of 64.38% which outperforms a random classifier but not greatly. They discuss at length the likely causes of the results, stemming from the data quality and formalisation as limitations to the Strange Beta routes and the CRDL used to transcribe them – like the downsides mentioned previously. Because of the small dataset, they combine two different grading scales which will add noise to the data – leading to badly presented grades for the routes.

Using the MoonBoard once again, Geary and Valdez, (2019) propose using an Auxiliary Classifier Adversarial Network (ACGAN) to generate climbing routes of a desired grade for the application. This project remains unfinished, so they have yet to finish creating a stable generator. They plan to complete their work by adjusting the hyper-parameters to account for the discrete differences between the grades and conduct a user study with surveys. The final product would be a fully functioning system for climbers on the go, with an application or website and interactive community.

Stapel, (2020) proposes a greedy algorithm using heuristics based on the analysis of the individual holds on a MoonBoard. This algorithm will generate routes at a standard comparable to human setters. Experienced climbers completed a Likert survey to rate the usability of every hold on the board, this rating formed the heuristics used in the algorithm – something previous literature has mentioned as further work to improve their results. Stapel reflects on the work of Phillips et al, (2011); Dobles et al, (2017); and Kempen, (2018) commenting that by combining his work with the ML techniques from other literature, a classifier can accurately determine the grade of a route.

Much like the work of Dobles et al, (2017), graph convolutional networks (GCN) are applied by Tai et al, (2020) to the MoonBoard apparatus to classify grade routes. However, they apply the GCN in conjunction with multi-hot feature embedding achieving an average AUC score of 0.73 – much better than classic statistical learning algorithms and the CNN implemented by Dobles et al, (2017) which reached 34% accuracy. This implementation uses a custom script to strip the metadata tags, labels, and features instead of the usual image classification in other literature. They find the best results occur with four convolutions, a discovery that deviates from the conclusion by Yao et al, (2019).

Lo, (2020) employs a variational auto-encoder to the MoonBoard environment to generate climbing routes of a high quality. Before training the auto-encoder, they randomly split the routes to ensure that problems with various style and difficulty are fed into the algorithm – this resolves any risk of bias. Once trained, 50 routes are generated and manually validated to filter out routes that are un-completable, too short or missing start/end holds – leaving only high standard and well-flowing routes. The downsides to this study are that only 22 of the 50 generated routes are viable. Meaning the routes must be manually verified to return less than half the generated total. Additionally, the 22 routes outputted are labelled as "high quality"; but nowhere does it state the measure of quality or difficulty, it is purely opinion.

An alternate method to estimating climbing route grade from Scarff, (2020), is by using Whole-History-Rating (WHR). Adapting the WHR model to climbing routes, the researcher is able predict a route grade with 85% accuracy with 10-fold cross-validation. Data was collected from *theCrag,* a website used for climbers to log their ascents. A sample was pre-processed to remove ambiguous and un-graded ascents. The disadvantage to this data source is that the model relies on users of the website to accurately record their climbs. Hence, looping back to Draper, (2016) the data relies on subjective factors. A downfall of the model itself, is the output grades cross multiple boundaries – making it unclear the definitive rating. This project has interesting possibilities, because it is common for old, graded routes outdoors to change difficulties when re-climbed.

Finally, Duh and Chang, (2021) introduce BetaMove, a new move pre-processing pipeline utilising Recurrent Neural Networks (RNNs) for MoonBoard route classification and generation. The grade predictor reaches near human-level performance and the generator produces routes of better quality compared to previous literature Houghton et al, (2020). They intelligently pre-process the data to remove any ambiguous such as routes that are from amateur users or mislabelled. This prevents some of the subjectivity of the grading and prevents any bias. Moreover, they pioneer RNN architecture for route classification and compare with other classifiers completed by previous literature. This study puts forward a promising pipeline for future work and transfer learning in this field.

2.1 Summary of Literature

In conclusion, Philips et al, (2011) were the first to explore this field and lead to all other studies mentioned. The current best performing classification for climbing routes on the MoonBoard is presented by Tai et al, (2020) with discussion of potentially developing a generative solution to MoonBoard routes, much like the work of Geary and Valdez, (2019); Stapel, (2020); and Lo, (2020).

2.2 Similar Projects

Dobles et al, (2017) were the first to apply ML methods to climbing routes - specifically the MoonBoard. They apply three models to the whole MoonBoard dataset, including a CNN across 13 classes whereas, this project is across 3 classes with 3000 routes to avoid the issues discussed in Section 4.1. Similarly, Tai et al, (2020) used GCNs then Duh and Chang, (2021) use RNNs. A GCN and RNN could have been applied in this project to compare which method best classifies however, the dataset format would have to be altered – more on this in Section 4.1.

Chapter 3

3 Methodology

3.1 Project Management

3.1.1 Gantt Chart

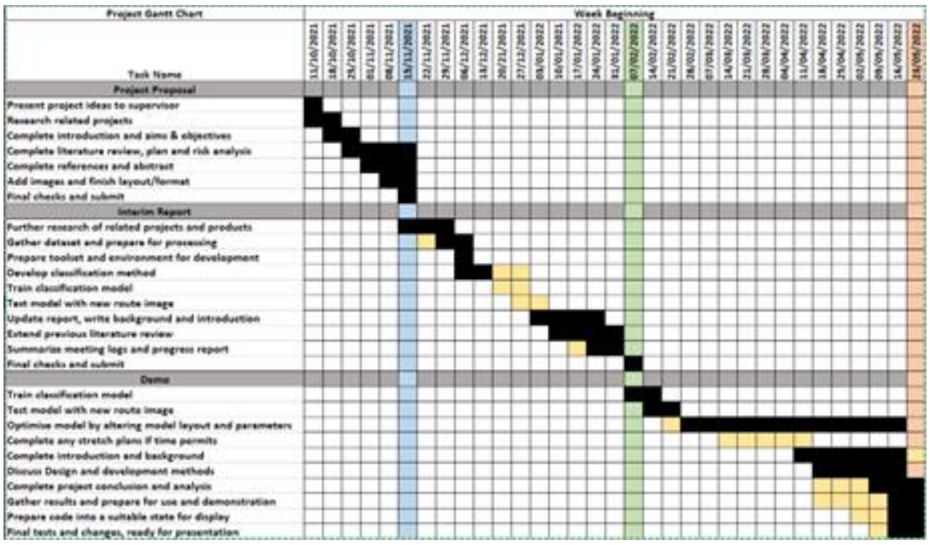


Fig. 1. *– Gantt chart updated from project proposal and interim reports to show the progression and planning of the development.*

Gantt charts are used in this project for planning and management. Each section displays the tasks required to complete the project, with the estimated time (yellow) and actual time (black). This shows an overview of the tasks to development, further discussion can be found in Section 4.2. Gantt charts are used for the capability to visually track progress from week to week (Gilmore, 2016). Due to the time constraints and challenges of this being a solo project, a Gantt chart ensures all tasks are completed on time.

Since the initial proposal, the Gantt chart has been updated to accommodate any new tasks necessary to log as they occur during development. Because this project will involve learning new topics, it is vital to prepare for unforeseen implications and fitting those into the timeline. These updates were made because not all tasks were completed in the same number of sub-tasks or in the projected time-period. The chart also shows where some tasks took longer than expected and bled into the allotted time of other tasks. E.g., to train and test the classification model took a short amount of time but optimising took more than double the planned time.

To summarise, the Gantt chart was useful for logging the project progress by providing a visual list of tasks to complete and the time needed. Without the chart, a basic list would not provide the same detail of plan; causing the project to fall behind schedule and miss integral tasks. (Chadwick, 2022).

3.1.2 Supervisor meetings

An important part of the project management was the help from the supervisor. From the beginning meetings were held to discuss the aims and scope of the project, then continued to chart progress and solve queries or issues. Throughout the project the supervisor has provided the guidance needed to complete each stage. They kept all work on topic and ensured sufficient progress was being made with each meeting. They also provided a second opinion and pair of eyes, helping to avoid common pitfalls and bring more experienced consideration to the discussion.

Overall, the guidance from the supervisor was integral to the outcome of this project. They verified the validity and relevance of the project to be a meaningful and worthwhile study.

3.1.3 Risk Assessment

The risk assessment of this project is ported over from the proposal. The only changes made are the dataset risks mentioned in Risk No. 4, this is no longer applicable because the MoonBoard dataset was selected for this project. For the proposal the dataset was not decided yet, and the MoonBoard data is free to use; with no risk of it changing permissions or disappearing. During the development, no potential risks occurred as expected. The only risk that almost became a problem was No. 3 but with TensorFlows' (Abadi et al, 2015) functionality of GPU utilisation and Colab (Google, 2017); this was avoided.

Risk No.	**Specific Risk Details**	**Likelihood**	**Assessed Impact**	**Manage/Mitigation Action**
1	Current open-source software and resources changing permissions / becoming proprietary.	Low	High	Keep a version history of resource use so it's available to be implemented.
2	Changes in government polices / ethical guidelines.	Low	Medium	Only a few policies / guidelines have effect on this project, should changes occur the project can be altered accordingly.
3	Unexpected lack of computational sources.	Low	Medium	As I have not reached that stage of development yet, it's unclear as to what computational resources are required. Should extra resources be needed I can use the lab facilities.
4	Absence of sufficient datasets	Medium	High	Gathering the data could be done manually or augmented. Additionally, training tools like the MoonBoard and Kilter Board are a possible data source.

Table 1. *Risk assessment table for entire project.*

3.2 Software Development

3.2.1 AGILE

For this project the AGILE development strategy was used, in combination with SCRUM. AGILE is deployed because of its flexibility, which affords the necessary freedom to account for unforeseen aspects of development. This benefits from more frequent feedback about the current progress of the artefact (Paterska, 2021) and breaking the code into smaller more manageable sections to iteratively test and reach more goals (Vijayasarathy and Turk, 2012). For tuning and hyper-parameterisation of the model, the iterative nature of AGILE was especially useful for testing values when the processing time was unknown between runs. AGILE best fits the roadmap of the project because tasks are likely to change, and this strategy permits changes without disrupting the rest of development (Forbes, 2016). AGILE is also a risk-reducing framework, since development is split into segments with testing; any errors that would normally accumulate will be resolved quickly (Diebold and Mayer, 2017). Since plans can be changed effectively, when any new ideas to improve existing code arise; it can be implemented seamlessly to achieve a higher quality end artefact (Forbes, 2016).

To summarise, the AGILE framework for software development was effective for this project by giving the freedom to make mistakes and correct the plan without effecting the rest of the artefact. Other frameworks would be too restricting or require additional planning that would detract from other aspects of the project (Forbes, 2016).

3.2.2 SCRUM

The meetings that occurred with the supervisor act as a SCRUM framework, where the tasks set between meetings are sprints. SCRUM compliments the AGILE framework well because sprints can be altered when necessary if something changes in the plan or meetings (Layton, 2016). Additionally, by planning the development in segments it allows the backlog of tasks to be completed with different levels of priority (Chandana, 2022). For a solo project the combination of AGILE and SCRUM is the most suitable development strategy to plan and track progress; in an adaptable environment to ensure the best product in the given time.

3.2.3 Ethical Consideration and Forms

For this project there are no ethical issues to consider or avoid, because no human participation is involved in the development or testing of this artefact. Hence, an EA1 form is completed and attached. The choice to not utilise human participation in this project was because it was not necessary for the scope.

3.3 Toolsets and Machine Environments

For the development of this project, Python 3.10.1 was used. This is because of its stability, flexibility, and range of available tools (Kumar, 2021). The benefits and drawbacks to this programming language will be discussed in the context of this project in this section.

3.3.1 Libraries and Frameworks

The main benefit to Python is the available libraries to utilize. A major library used is TensorFlow (Abadi et al, 2015) with Keras library built in. TensorFlow is a popular library for deep learning with extensive documentation and functions to import. An alternative to TensorFlow is Scikit-Learn or Sklearn (Pedregosa et al, 2011), which is intended for general ML problems. Both libraries vary, Sklearn comes with off-the-shelf modules that are highly abstract and can be implemented with minimal coding. Whereas TensorFlow provides the functionality to build and configure models with more freedom but at the sacrifice of efficiency with batching and standardisation (Bajpai, 2020). TensorFlow is better suited to complex neural networks making it more suitable for this project (Gupta, 2021). With the freedom to build models however desired, parameters can be altered with testing and lead to significant performance changes. For this project models can be built around the dataset such as adding convolution layers for CNNs and tweaking the number of neurons. Additionally, pre-trained models can be imported for transfer-learning such as VGG (Simonyan and Zisserman, 2014). With Sklearn the data must be processed beforehand whereas TensorFlow will automatically extract valid features from the dataset (Bajpai, 2020). E.g., Keras features ImageDataGenerators, which allow for image processing and augmentation at runtime to save memory space when models are training. A downside to TensorFlow is the slower operation speed on

CPUs, compared to other frameworks. Examples of TensorFlow being slower than Sklearn are discussed by users: Cyb70289, (2017) and Pplonski, (2020). Additionally, Shi et al, (2017) benchmarks deep learning tools and shows that TensorFlow performs worse on CPU and GPU. Despite the slower performance, TensorFlow provides optional GPU and TPU utilization to significantly increase operation time compared to Sklearn, which does not have GPU-accelerated operations (Bajpai, 2020). Since the dataset for this project is images, it takes an considerate amount of computation power and time to train each model – making GPU utilisation a necessity to quickly test and reconfigure models.

Other notable packages available to Python that are used in this Project are: Matplotlib (Hunter, 2007), Seaborn (Waskom, 2021), PIL (Lundh, 2011) and OpenCV (Bradski, 2000). Matplotlib is a visualisation library for figures, and Seaborn builds on top to provide extra functionality, in this project these libraries were used to plot model performance, either as graphs or confusion matrices. PIL and OpenCV are used for image processing. Other libraries less used in this project are NumPy (Harris et al, 2020) and Pandas (McKinney, 2010), for data analysis and matrix manipulations in conjunction with Matplotlib/Seaborn to output results. All these packages are actively maintained with detailed documentation and community usage, including major companies such as Google, IBM, and Intel. Alternatives to these packages exist but are not the made to the same industry standard.

Justification for using Libraries

For the scope of this project, where necessary, a library was imported to fulfil a function however, the choice to code from-scratch or import is a topic of discussion for software engineering as shown by these forum examples; ChssPly76, (2009); Ahmed, (2018); and Cowpig, (2018). The choice should be made by considering the scope, cost, and time of the project management i.e., The Iron Triangle (Barnes, 1969). Nikola, (2021) discusses the factors to decide between libraries or from-scratch by applying the concept by Barnes, (1969) to Python, these factors can be applied in this project. The main factor to consider is the time constraints, when attempting tasks that fulfil the research purpose but not become too ambitious. Hence, the use of libraries in this project, developing without these frameworks would take a significant amount of extra time.

A second factor is the scope of the project, since this is a solo project with high difficulty, it is worth taking options to simplifiy where possible. Additionally, this project will not be deployed to production but could be made open-source through GitHub. If other developers were to access the code, they would have a harder time working from a scratch implementation than one using a library; since the library code is widely recognised and documented. Building from-scratch means needing to understand much more about the code and principles of the function you are trying to program – resulting in more problems and extending the difficulty to beyond the scope of the project. E.g., neural networks like the CNNs in this artefact would be very complex to build from nothing without a library framework (Nikola, 2021).

The last factor to consider is the cost and use of resources. A downside to using libraries is that it adds dependencies to the software however, since this project will not be deployed for production; dependencies are not an issue. Another downside is that a large library will contain unused functions, wasting unnecessary resources. To resolve this, certain functions from a package can be selected and imported. Since Python and the packages utilised are free open-source, there is no monetary cost to consider.

In summary, libraries provide a tried and tested code-base to reuse; eliminating the need to write the code from scratch and run into potential problems. Koert, (2018) supports this by mentioning that frameworks and libraries have stood the test of time so why reinvent the wheel.

3.3.2 Popularity, Flexibility and Readability

In recent years Python has rapidly increased in popularity and use to become the go-to language for ML, as shown by queries on Stack Overflow (Robinson, 2017), contributions on GitHub (Elliot, 2018) and job postings on Indeed (Puget, 2017). Due to this popularity, the longevity of Python projects is increased; as the number of resources will continue to grow and be maintained, leading to more resources to learn. As a result, Python is used in this project because of its popularity for ML tasks, range of resources and future-proofing potential. Alternatives to Python and how they compare are discussed later in this section.

A final positive to Python is the flexibility and readability. It is a high-level language; with a low entry barrier, making it more accessible for developers (Dialani, 2021). This ease of use makes Python a simple language for testing and prototyping problems - especially for ML (Patel, 2018). Python offers flexibility to use different styles of development e.g., object-oriented or scripting and the versatility of different platforms like Linux or MacOS; for testing and deployment (Luashchuk, 2019). With Python, the robust code will avoid complications, and any work by other developers can be simply implemented should this project be made open source. To summarise, it is because of these reasons that Python is used in this artefact and many other ML or deep learning projects.

3.3.3 Alternate Languages

Although Python is the current leading programming language for ML, Python has some downsides and alternate languages should be considered. Python scales poorly to the computational demands of ML and make it difficult to provide model-level parallelism (Innes, et al 2018). The next alternative to Python is C, which is a faster language for deployment and less computationally taxing (Patel, 2018). However, C is more complex in syntax and de-bugging than Python; making it slower for development when additional knowledge and detail for prototyping and testing is required (Sidana, 2021). “An ideal ML ecosystem is an ideal numerical one” (Innes, et al 2018, 2) with this argument, the Julia programming language (Bezanson, 2017) was authored. The faults with Julia are the instability, lack of an IDE, and limited number of third-party packages (Gao et al, 2020). The main difference between Python, C, and Julia is that Python is an interpreted language whereas the other two are compiled. Compiled languages are faster performing but cannot be modified while running, and all debugging occurs at runtime rather than prevent the code from compiling (Shubhamsingh10, 2021).

3.3.4 Environment

The environment or IDE for this project was Google Colab, which is like Jupyter Notebooks but hosted on Google’s servers and resources. With Colab, Python can be written and executed utilising Google’s computational resources including GPU and TPU to significantly increase processing time for deep learning models. By using Google’s resources there is no problems that occur when trying to optimise for local systems. Many popular libraries and frameworks are already included, projects are hosted on Google Drives so can be easily shared with other developers (Google, 2017).

Chapter 4

4 Design, Development and Evaluation

4.1 Requirements Elicitation, Collection, and Analysis

4.1.1 Dataset

The dataset for this project is from the MoonBoard climbing wall, sourced from (Moon Climbing, 2019). The MoonBoard is a standardised climbing tool available in numerous climbing gyms globally, every MoonBoard is the same setup where users can create and load climbing routes from a mobile application. A route is a subset of the holds signalled by coloured circles on the application and LEDs on the actual board, with starting, intermediate, and ending holds. See Figure 2.

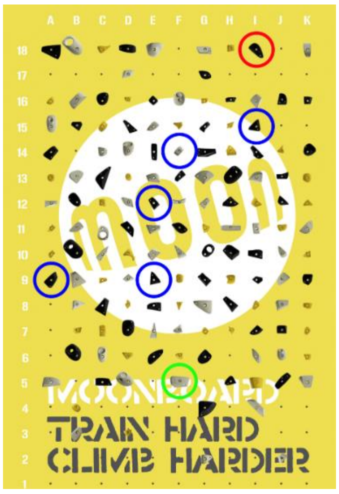


Fig. 2. *– Example route used in the dataset of a MoonBoard route. Green circles represent starting holds, blue shows intermediate and red signals the finishing hold (Moon Climbing, 2019).*

The hold set used in this project is the original 2016 set, which is an 18x11 grid of 142 holds. This hold set is used because it is the oldest and therefore, there are more available routes with more repeats and ratings, meaning the routes collected for this project are of higher quality and less susceptible to debate on its grade.

For this project, 3000 images are collected, while there is a lot more routes on the application to use, this sample size is just a proof of concept. Additionally, the more images used the more susceptible for low quality or miss-graded routes; that are not reviewed by the community. If the whole collection of routes was used the distribution of grades would be severely skewed, because most routes on the MoonBoard are in the lower range of grades – this problem is mentioned in other related literature. It is worth noting that the skew in the lower grades is because most climbers are at that skill level therefore, more routes will be created and attempted at that level. This supports the decision to use the whole dataset by Dobles et al, (2017); Tai et al, (2020); and Duh and Chang, (2021) as the skew is a natural bias created by the climbing community but must be heavily counterbalanced in their study to achieve relevant results from ML models. With the selection of 3000 routes, each class is evenly distributed hence it represents each climber skill level evenly and without the need for balancing that potentially affects the performance.

When collecting the images, they were sorted by most repeats and filtered by three groups of grades for each class. 6B+ to 6C (Easy), 6C+ to 7A (Medium) and 7A+ to 7B (Hard). Grades above these ranges are not collected because the number of available routes is so small and routes that highly graded are extremely difficult, with only a few climbers in the world that could climb them. Additionally, because of the high grade it is likely they are graded higher than they are, since users can assign the grade of a route they create to whatever they want (Walker, 2020). See Figure 5 for the filtering screen.

The MoonBoard is used for this project because of its standard design and wide catalogue of routes for training ML models. The design of the MoonBoard is what makes this project feasible by classifying routes within the constraints of the wall. Because the MoonBoard holds are the same in every route – including the orientation and distance between them, the board can be modelled as a two-dimensional plane as opposed to the three-dimensional walls and routes used in regular indoor climbing. Even though the routes use variations of the same set of holds, the number of potential routes vastly expansive. To apply the model to grading routes on indoor circuits with varied holds and wall angles is possible but extremely difficult, because the possible configurations of a route are endless. Hence, the use of the MoonBoard in this project and other projects from referenced literature.

However, the MoonBoard problem space does have some downsides. Since the wall is flat and overhanging with no features or volumes, the climbing styles are limited and focuses on strength and neglects technique, balance, and flexibility (Walker, 2020). The other downside is with the format of the dataset itself from the MoonBoard application. Other literature projects that utilise the MoonBoard either use scraping scripts to extract metadata tags, labels and features or heuristics of the holds. This project uses the images from the application directly, as seen by any user. The problem with this is the stock images of the MoonBoard contain a lot of noise i.e., the background logo; that is likely to affect the model performance (Lau et al, 2021).

4.2 Design

As discussed, the purpose of this project is to classify MoonBoard routes with a CNN. The steps planned in the design and development of this artefact are as follows:

1. Collect dataset
   a. Filter routes on MoonBoard application
   b. Split into classes
2. Pre-process dataset
   a. Crop down to uniform size without irrelevant information
   b. Copy versions in grayscale and without background
3. Split into train, validation, and test sets
4. Train model
5. Evaluate and test model
   a. Evaluate and test for each dataset
6. Optimise model
   a. Test different epoch size
   b. Test different batch size
   c. Test different CNN structures by changing layers and neurons
7. Retrain model with new parameters
   a. Evaluate and testing of new architecture

4.2.1 Dataset Pre-processing

As mentioned, the images were collected from the MoonBoard application (Moon Climbing, 2019). These were cropped down to just show the route setup and remove the irrelevant user interface from the image. It was cropped using the PIL Python package, to iteratively crop the images and overwrite the original on disk ready for further pre-processing. To test how the dataset affects the accuracy of the model three versions were created; one original set, one grayscale (labelled: GS data), and one without the yellow background (labelled: No BG data). To convert to greyscale, OpenCV was used to convert the colour space from RGB to grey – see Figure 3. To remove the background, PIL was used to loop through the height and width to change the RGB values matching the background to white – see Figure 4. Since the holds are a different hue to the background these are not affected. Using grayscale and without the background data, the images contain less information to potentially affect the model performance i.e., image misclassification caused by morphological similarity or non-essential information interference (Bennet and Min, 2020).

Bennet and Min, (2020) describe morphological similarity as the innate inter-class similarities in structure, for this dataset this misclassification is likely because each MoonBoard image is identical other than the circled subset of holds marking the route. *Ibid* define non-essential information interference as a phenomenon where pixels outside of target features dominate the components to classify. This is probable for when the circled holds are near other holds not in the route, leading to confusion for the model to distinguish the defined route from the rest of the board. Similarly, this interference can occur between the edges of the hold and background; particularly around the logo watermark, it is possible the background dominates the boundary to where a hold or circle begins and ends and thus lead to misclassification. This phenomenon is analysed in Section 4.4 when filter size is tested.

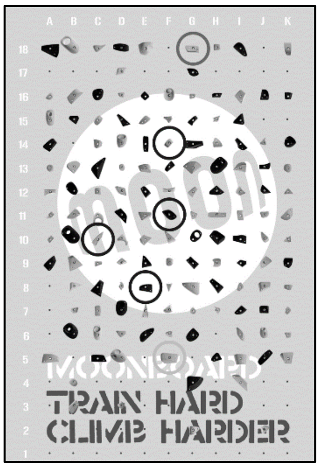


Fig. 3. *– Sample greyscale data.*

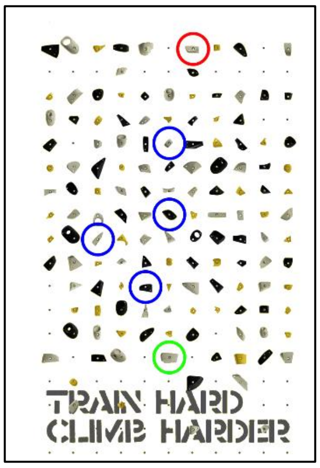


Fig. 4. *– Sample data with background removed.*

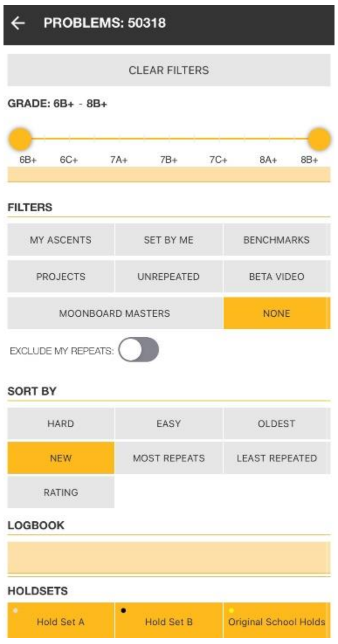
To split the data into training, validation, and testing; the split-folders function was imported to partition each dataset into 70:20:10 splits with the same seed. To increase the size of the dataset, augmentation was considered since the size of the dataset affects the performance of the model (Brownlee, 2016). However, it was not used because altering the data with augmentation detracts from the standardised nature of the MoonBoard. Augmentation such as random shifting, flipping, or rotation of an image would lead to a route that does not belong to the MoonBoard problem space and likely lead to confusion or inaccuracy in classification. Additionally, data augmentation leads biases compared to the original dataset (Xu et al, 2020), and finding the optimal augmentation solution is a difficult challenge (Soni, 2022).

A component considered in this development is the image input size fed into the ML models. Some models used for transfer learning are expecting certain sized images and/or have been trained initially with these sized inputs – such as VGG was trained with 224x224 images. However, at smaller resolutions that image loses quality and therefore important features to the training of the model and classification output. The problem with higher resolution is that the models will take longer to train and transfer learning model performance might suffer at higher input sizes.

For the images in this project, lower dimensions squash the image terribly which will lead to incorrect classifications. The full image size is 827x1215 (after cropping), for the model input size they are resized to 400x600 to retain the aspect ratio and features but cut the training time down as much as possible (Luke et al, 2019).

3000 images are collected separated into three classes (Easy, Medium, and Hard) with 1000 in each. It is worth noting that the class names do not necessarily represent the difficulty of a route, because the opinion of what an “easy”, “medium” or “hard” route is different for every climber. For this project the class names simply represent the three ranges of grades, especially since the MoonBoard is a training tool for more experienced climbers.

For the ML models of this project, the dataset is split into a 70-20-10 split of training, validation and testing sets respectively where each split contains the same random selection of the three classes each. The training set has 2100 samples, the validation set has 600 and the testing has 300. This is to ensure the model does not memorise the data additionally, the validation and testing sets are for evaluating the trained model on new unseen data and view the best performing model or model layout such as the number or layers or convolutions etc. (Landup, 2022). To obtain the classes of images, inside the application the catalogue of routes can be filtered to specific grades or popularity – see Figure 5.


PROBLEMS: 50318

CLEAR FILTERS

GRADE: 6B+ - 8B+

6B+ 6C+ 7A+ 7B+ 7C+ 8A+ 8B+

FILTERS

MY ASCENTS | SET BY ME | BENCHMARKS

PROJECTS | UNREPEATED | BETA VIDEO

MOONBOARD MASTERS | [✓ NONE]

EXCLUDE MY REPEATS:

SORT BY

HARD | EASY | OLDEST

[✓ NEW] | MOST REPEATS | LEAST REPEATED

RATING

LOGBOOK

HOLDSETS

[✓ Hold Set A] | [✓ Hold Set B] | [✓ Original School Holds]

Fig. 5. *– Filtering screen from the MoonBoard application, to show specific routes of the user's choice (Moon Climbing, 2019).*

4.3 **Building and Coding**

4.3.1 File Structure

The code base for this project is built in two scripts Project.ipynb and Image_Processing.ipynb. In this subsection the two files will be described for their purposes and operation.

Image_Processing.ipynb

This script is where all dataset pre-processing takes place before training. This file houses five functions run separately, these functions are for: cropping, converting to greyscale, removing the yellow background, resizing images, and for splitting into training, validation, & testing sets. The outputs of these functions are the modified images stored locally for processing. This code is written in a separate file from the main code to keep the two parts of development independent to add modularity to the artefact. Any new data to test or predict with the main code can be pre-processed in this file to be the same format.

Project.ipynb

This is the main code file for the whole project, it is here the datasets are trained and tested across the multiple ML models. Firstly, the dataset is loaded from the directory using ImageDataGenerators, the different datasets can be loaded by switching the read path. Secondly, all models are trained, tested, and evaluated separately, the results are collated and graphed. With this code, any new MoonBoard route images can be classified, and display performance metrics.

4.3.2 Discussion of Models Implemented

For this project as mentioned, it will be implementing a Convolutional Neural Network (CNN) to classify the MoonBoard route images into three classes. A single function is created that takes the arguments: training & validation data, target size, classes, batch size, and number of epochs. A simple network was implemented following the article by Sarkar, (2019), along with several pre-trained models as transfer learning to compare. Torrey et al, (2006) describe transfer learning as a mechanism to increase learning speed and performance by avoiding data-labelling and re-training (Tang et al, 2018) hence, the use of pre-existing models in this project. Prebuilt deep learning models are used in comparison for their award-winning performance on image classification and to test a simple architecture with complex very deep architectures. The architecture of the basic CNN is shown in Figure 6.

CNNs are used for this task for their performance and popularity for image classifications (Boesch, 2022). With the recent progress in CNNs displayed by Krizhevsky et al, (2012) of AlexNet; Simonyan and Zisserman, (2014) of VGG; He et al, (2016) of ResNet and more (paperswithcode, 2022), CNNs continue to improve in performance and use across numerous applications of image classification. E.g., Maruyama et al, (2018) prove the effectiveness of CNNs in the case of medical image classification. Furthermore, Tang et al, (2019) builds on top of a CNN for image classification with large variation.

The benefits of CNNs compared to other ML models for image classification are because of the sliding filter in the convolutional layers. This filter segments a piece of the image at time which aggravates the dimensionality (Mishra, 2019). CNNs does not require hand-crafted features for the input, which eliminates the need for feature selection saving time in development and avoids user bias (Al Zorgani and Ugail, 2018). Compared to other ML models the performance for image classification, CNNs consistently score higher than alternatives as shown by Hasabo, (2020). Moreover, versions of CNNs continue to win image classification competitions such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) since 2012 (Sharma, 2020). It is because of these reasons that CNNs are utilised in this project.

The advantages of using pre-trained models (PTMs); like the ones implemented in this project are presented by Han et al, (2021). Use of PTMs can achieve great performance for labelled or unlabelled data without having to re-gather the knowledge stored for huge amounts of parameters and fine-tuning. These PTMs can benefit a wide range of tasks when altered (*ibid),* even major frameworks like TensorFlow, Keras and Fast.ai offer pre-trained models which are used by many major companies as mentioned in Section 3.3 (Shao, 2019). On the other hand, He et al, (2018) challenge the paradigm of 'pre-training and fine-tuning' with experiments of comparing models with random initialisation to pre-trained ImageNet counterparts and showed no loss in performance. Results show that pre-trained models converge faster in training but do not necessarily achieve higher accuracy (ibid, 8). Hendrycks et al, (2019) agrees with He et al, (2018) that pre-trained provide no performance increase but counters by proving that they do improve model robustness and uncertainty estimates. With these points in mind, pre-trained models offer more benefits than drawbacks as mentioned by He et al, (2018) and Hendrycks et al, (2019) which apply especially for this project, the main benefit is that they enable improvements for small-scale datasets (Zeiler and Fergus, 2013). Secondly, they "reduce research cycles leading to easier access to encouraging results" (He et al, 2018, 8).

To summarise, a simple CNN and a handful of pre-trained models are implemented in this project to compare the performance for classification on MoonBoard climbing routes. The CNN will provide a baseline performance to compare with the models from transfer learning. Initial testing will reveal the optimisations required for the CNN to be altered and re-tested.

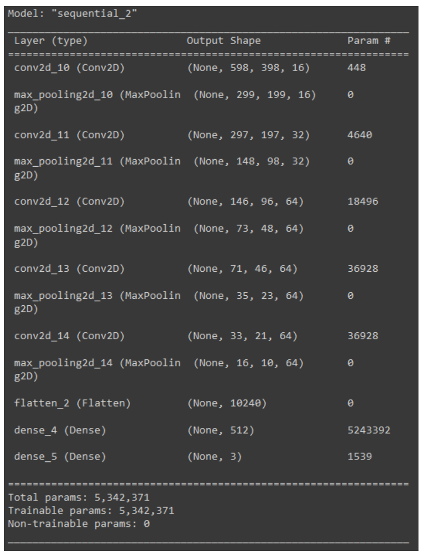
4.3.3 Environment, Dataset and Model Preparation

The first step to building this artefact is preparing the environment for development, a new Colab Python file was made, and all pre-processed datasets were uploaded to Google Drive for access.

Once the environment is ready the pre-processed dataset needs to be prepared for the ML models. The data is formatted accordingly to the documentation for training and testing. Three ImageDataGenerators for each set were created to feed the images into the model at runtime. It is at this point the images are resized to 400x600 belonging to their respective classes. The simple CNN function takes the training and validation generators, steps per epoch, number of epochs, batch size and classes as arguments to the model.fit function – the same arguments are passed when fitting the pre-trained models.

4.3.4 Model Architectures

The simple CNN features 5 convolution layers whereas the PTMs have much more for greater depth – shown in Table 2. Following the convolutional layers in the CNN, the data is flattened into one-dimensional array leading to a single hidden layer; of a dense neural network with 512 neurons and finishing with 3 neurons on the output layer for each class, using an RMSprop optimiser with a 0.0001 learning rate. The model is trained for 30 epochs at a batch size of 32 – these initial parameters are to ensure classification is possible regardless of the performance, any altering of the hyper-parameters will be tested later for optimisation. For each PTM, the convolutional layers are imported with the ImageNet weights with the output dense layers altered for the classes of this dataset. As mentioned before, these pre-trained models are trained on images of size 224x224 or 299x299, so for the images in this implementation the performance of these models might be affected (Richter et al, 2021).



```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 598, 398, 16)      448

 max_pooling2d_10 (MaxPoolin  (None, 299, 199, 16)     0
 g2D)

 conv2d_11 (Conv2D)          (None, 297, 197, 32)      4640

 max_pooling2d_11 (MaxPoolin  (None, 148, 98, 32)      0
 g2D)

 conv2d_12 (Conv2D)          (None, 146, 96, 64)       18496

 max_pooling2d_12 (MaxPoolin  (None, 73, 48, 64)       0
 g2D)

 conv2d_13 (Conv2D)          (None, 71, 46, 64)        36928

 max_pooling2d_13 (MaxPoolin  (None, 35, 23, 64)       0
 g2D)

 conv2d_14 (Conv2D)          (None, 33, 21, 64)        36928

 max_pooling2d_14 (MaxPoolin  (None, 16, 10, 64)       0
 g2D)

 flatten_2 (Flatten)         (None, 10240)             0

 dense_4 (Dense)             (None, 512)               5243392

 dense_5 (Dense)             (None, 3)                 1539

=================================================================
Total params: 5,342,371
Trainable params: 5,342,371
Non-trainable params: 0
_________________________________________________________________
```

Fig. 6. *– Simple CNN architecture summary showing each layer and the number of neurons.*

4.4 Testing

Testing in this project was a combination of unit and white box testing. White box testing is used to test the artefacts' underlying code to ensure efficiency, unit testing is implemented to validate the functionality of the code to guarantee it works with each new block of code (Perfecto Team, 2022). General performance testing is used to view the performance of each model. The merit of using a cell structured environment like Google Colab, is that each cell can be tested piecewise without running the entire code (Yalçın, 2020) – making development and testing much quicker and easier (Sjursen, 2020), which act as simple unit tests.

The reason these testing strategies were used because of their effectiveness at testing the whole functionality of the software to ensure the result is reached on time and performs as expected. White box testing resolves any bugs that might be missed by other methodologies, it is a must for an efficient and high quality artefact (Dhaduk, 2021). Unit testing makes debugging faster and easier, code faults can be tested and changed without disrupting the existing code (Mikhalchuk, 2020). Additionally, they help gauge performance of the software which saves resources searching for issues later because it can be backlogged through the unit tests (*ibid)*.

To test the Image_Processing.ipynb file each function is run to save the altered image in a new directory and validated, if the output was wrong; they can simply be deleted, and the function can be debugged to run again. To test the Project.ipynb file, TensorFlow's model evaluation and prediction functions are used to obtain accuracy metrics. Since the output layer of all models use the softmax activation function, a confusion matrix was created to display the probabilities of predicting each class. As mentioned in Section 4.2: Data Processing, three datasets are trained and tested in this artefact. The datasets are split with the same seed and ratios and applied to each ML model. Table 2 showcases the results of each model when trained and tested with each dataset.

It is worth noting that all results from testing are subject to change with each run due to random seeding or sampling i.e., running the same experiment twice would yield slightly different metrics, making it difficult to achieve reproducible results (ODSC, 2019). Randomness occurs in the batching, and weight initialisation in ML models. Despite this randomness is important because it is more likely to achieve better results since a fixed seed might be incorrect (*ibid)*. Finding the optimal seed would take extensive testing and using a fixed seed would not lead to drastically different results anyway since the performance of a model comes down to its architecture and hyper-parameters.

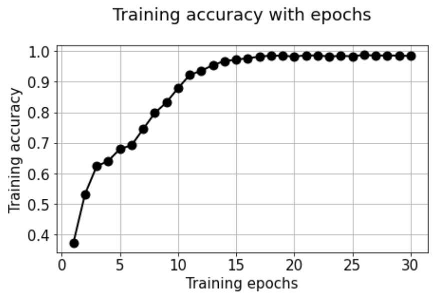


Fig. 7. *– Line graph of CNN training accuracy.*

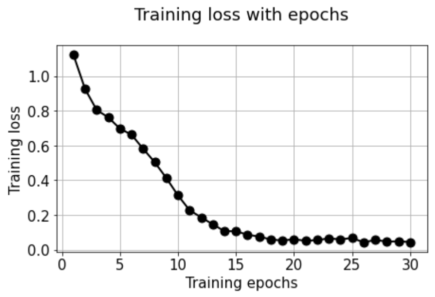


Fig. 8. *– Line graph of CNN training loss.*

These graphs show promising results but the fact that the training accuracy scored so highly in a few epochs could be a sign of over-fitting depending on the testing accuracy. Beyond 15 epochs the accuracy or loss does not change much therefore, in the optimisation of this model the performance can be tested with less epochs.

For testing the different pre-trained models, the number of dense layers and neurons was the same as the original with the output layer changed. If the model did not include a fully-connected layer, a single dense layer with 512 neurons was used to be the same as the simple CNN.

Model [Depth]	**Model Accuracy on each Dataset**								
	Train	**Val**	**Test**	**GS Train**	**GS Val**	**GS Test**	**No BG Train**	**No BG Val**	**No BG Test**
CNN [5]	0.989	0.5833	0.55	0.7071	0.615	0.6067	0.969	0.5783	0.6067
VGG16[1] [16]	0.5414	0.53	0.5167	0.511	0.485	0.5033	0.5271	0.495	0.5333
VGG19[1] [19]	0.5276	0.4750	0.51	0.5571	0.5183	0.5033	0.5390	0.5390	0.5390
Xception[2] [81]	0.6905	0.6	0.6167	0.6824	0.6017	0.6333	0.7143	0.615	0.6067
ResNet50[3] [107]	0.5857	0.5433	0.5567	0.3333	0.3333	0.3333	0.3671	0.365	0.3533
ResNet50V2[3] [103]	0.7919	0.6283	0.63	0.7376	0.605	0.63	0.6614	0.5667	0.5633
ResNet101[3] [209]	0.5895	0.5517	0.5867	0.4643	0.4867	0.4733	0.5114	0.5	0.51

[1] Simonyan and Zisserman, (2014)
[2] Chollet, (2017)
[3] He et al, (2016)

ResNet152[3] [311]	0.4929	0.4767	0.4767	0.53	0.5167	0.49	0.3367	0.3367	0.3367
InceptionV3[4] [189]	0.6048	0.5317	0.5667	0.679	0.6217	0.6	0.7081	0.5933	0.6267
InceptionResNetV2[5] [449]	0.4771	0.4633	0.49	0.639	0.5714	0.5967	0.6662	0.5717	0.5933
DenseNet121[6] [242]	0.5943	0.5367	0.6033	0.4957	0.45	0.4733	0.3429	0.34	0.34
EfficientNetB7[7] [438]	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333

Table 2. *– Model accuracies across multiple classification models, the depth of each model is in square brackets. The highest accuracy for each column is in highlighted in green, testing set scores are shaded.*

The results of each model show wide variation in performance, no single model outperforms the others for each category. The best overall performing model was ResNet50v2 which performed highest on the original and greyscale sets with the most consistent was Xception which never dropped below 0.6, but most models produce similar results within a small range. The depth of the model has inconsistent effect on performance; as deep models like EfficientNetB7 with a depth of 438 never managed to converge whereas, InceptionResNetv2 with 449 depth performed the best on the No BG set.

[4] Szegedy et al, (2015)
[5] Szegedy et al, (2017)
[6] Huang et al, (2017)
[7] Tan and Le, (2019)

4 of the 12 models scored higher with the greyscale and No BG sets, which suggests these models are better suited to less complex images. However, use of the different datasets has no great impact on performance other than all the ResNet models and DenseNet where the other datasets performed much worse than the original. From the results the performance gain with the No BG set is higher than with the greyscale, supporting what was hypothesised in Section 4.2 that less image noise yields higher performance.

Sometimes a model will never converge and get stuck at ~ 33.3%, which across the three classes is effectively picking one class at random. This could be caused by several factors such as the data, model depth, filter size, or number of nodes. E.g., some models use a filter size that varies on different layers of the network whereas, the simple CNN implementation uses a fixed filter size. The relationship between the model structure and hyper-parameters with the performance is unclear at this stage for the dataset of this project. Hopefully the experiments to optimise the simple CNN, it will become clearer which components can be improved to grant a performance increase.

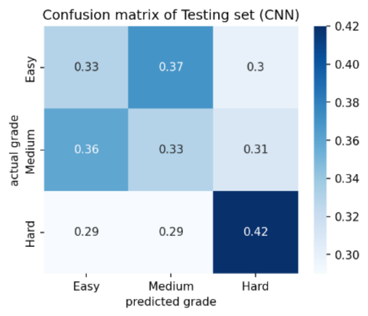


Fig. 9. *– Confusion matrix of CNN predictions on the original testing set.*

The confusion matrix shows poor prediction performance for all classes. The best performance is predicting a hard category but only with 42% probability. It is more likely to predict an easy class as medium than which defies the expected performance. The potential reason, is the class prediction fails for Easy and Medium, is because the data between them is probably too similar. Perhaps the subset of holds starts off the same and differentiate later in the route, so while they have different grades the routes are similar physically.

4.4.1 Optimisation

Following the results of the initial model performance, the next step was to optimise via tuning the hyper-parameters (Saragih, 2019). This section shows the steps taken to attempt optimizing the model by testing different values for different key hyper-parameters. Each subsection will discuss why the parameter was tested, the results of these tests, and an analysis of how effective the change was for the performance of the model. The testing to this optimisation took form as a trial and error approach of trying different values for each parameter, re-training, and re-testing / evaluating and graphing the results. However, this process was not completely manual, loops were used to execute the testing with each different value. This method for optimisation was used to test a range of potential values in a simple way that is not too time consuming but still thorough (Lord, 2021).

Batch Size

The first hyper-parameter that was tested was the batch size. Batch size in this scenario is the number of images processed before the model updates its weights. Batch size is tested because of the effects on performance as shown by Keskar et al, (2016), it is expected that with larger batch sizes; the model quality should degrade.

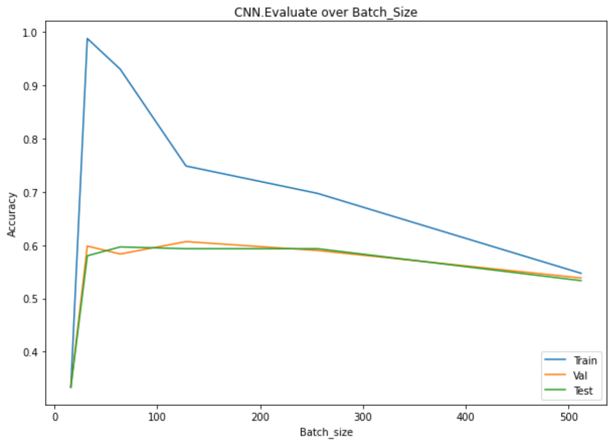


Fig. 10. *– Line graph of CNN model accuracy with an increasing batch size.*

The range of values for batch size used began from 32 and doubled up to 512, sizes below and above these values were not tested because of the limited dataset; it does not cover the required amount for the steps per epoch * epochs. Overall, the batch size had minimal effect on the validation and testing accuracies but generally show higher batch sizes perform worse. This drop in accuracy is expected as similar results can be seen by Masters and Luschi, (2018), Kandel and Castelli, (2020), and Keskar et al, (2017). Results show that larger batch sizes generalise poorly because they tend to converge to sharp minimisers in the training/testing functions whereas, small-batch methods converge to flat minimisers (Keskar et al, 2017). The reason the accuracy does not change much overall could be due to a few reasons, with this in mind the next parameter to test is epoch size to prevent any over-fitting and memorisation of the data. The highest testing accuracy occurs around the 32 to 64 batch size but the difference in performance is minimal, from this testing for future experiments the batch size will remain at 32.

Number of Epochs

The number of epochs is the number of times the dataset is passed through the network in the forward and backward pass. Since the dataset being passed is limited, to optimise the learning and obtain metrics to plot is an iterative process. Therefore, too few epochs will cause under-fitting but too many will cause over-fitting (Sharma, 2017). With the correct number of epochs, signs of over-fitting shown in Figure 7 should be resolved.

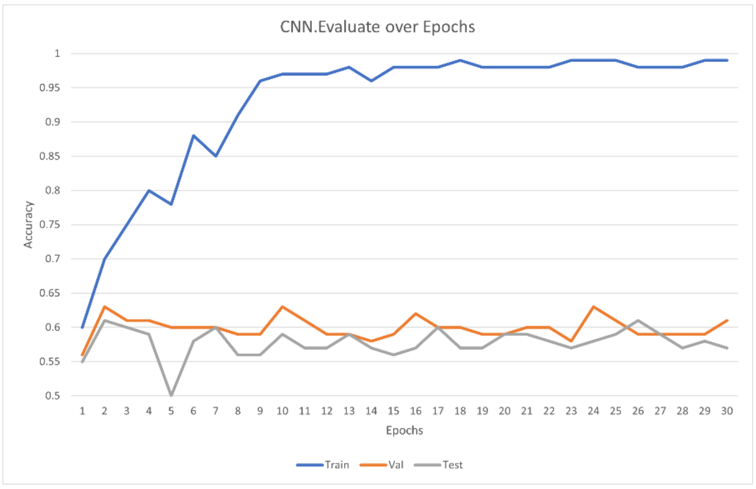


Fig. 11. *– Line graph of CNN model accuracy as the number of epochs increase.*

Epochs above the tested range were not tried because the training set stopped improving around 15 epochs and any further would risk the model memorising the data and over-fitting. The validation and testing lines show a lot of noise since the accuracy varies a lot for no real reason, this shows the model training is potentially unstable because the two sets should show similar trends since they are both unseen data. The testing score peaked at epochs 2, 7, 17, and 26; however, there is no increasing trend in accuracy between these values which defies the expected performance that when trained for longer the model is more likely to converge. To better show the performance of the validation and testing scores, Figure 12 shows the same graph with the training set filtered out and the y-axis limited.

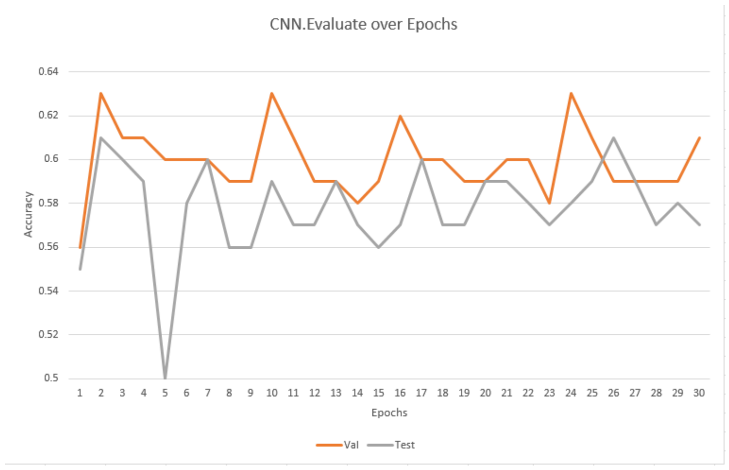


***Fig. 12.** – Line graph of CNN accuracy over number of epochs excluding the training set accuracy, same as Figure 11 but with the training set filtered out.*

Here we can better see the relationship of accuracy with increasing epochs. The cause of the noise is unknown; one potential factor is random seeding; if the test was running again, we can expect different results and see peaks in accuracies at different points. For the remainder of the optimisation testing, 20 epochs were used because 2 or 7 epochs is too few for the model to train properly – likely leading to under-fitting, also 20 epochs are just beyond the 3rd peak at 17 but less than 26 or 30 to save time and prevent over-fitting.

Optimisers

Optimisers are used to alter the weights across the network to minimise the cross-entropy loss continuously when training (Taqi et al, 2018, 142). The method which the optimiser does this depends on the optimiser chosen but the best option can vary for different tasks (Schmidt, 2021, 1) hence, all the optimisers within the Keras library are tested.

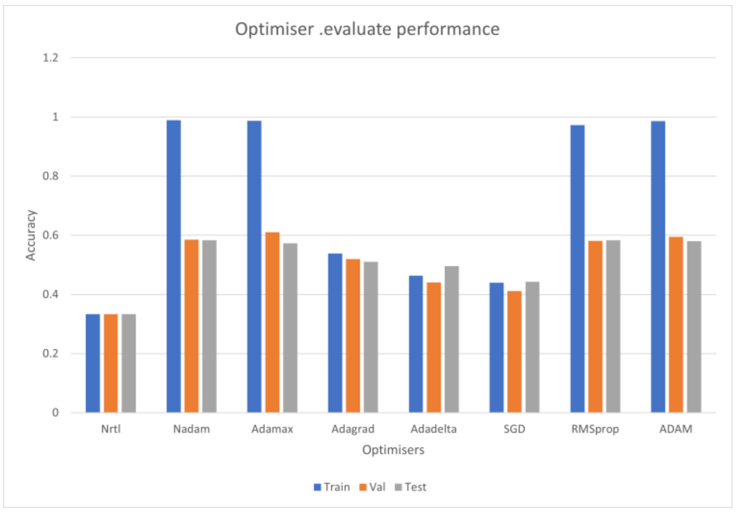


Fig. 13. *– Bar Graph of CNN model accuracy with different optimization algorithm.*

From the results it is clear which optimising algorithms perform best with Nadam, Adamax, RMSprop, and Adam achieving comparable accuracy scores for each dataset split. These results mirror the results from Bera and Shrivastava, (2020), Xia et al, (2020), Schimidt et al, (2021), and Taqi et al, (2018). The Adam optimiser consistently outperform other optimisers; with RMSprop following in second. Adamax displays slightly lower testing accuracy and higher validation accuracy, but this is likely due to noise or random seeding. For the remaining hyper-parameter testing any of the top performing optimisers could be used; however, effective CNN models like VGG or ResNet use the Adam optimiser. Therefore, the Adam optimisation algorithm will be used henceforth.

Learning Rate

An argument to the optimisers is the learning rate. The learning rate is the step size the optimiser adjusts the weights in the opposite direction to the gradient for a batch. With lower learning rates the training is more reliable; but optimisation will take more time because the steps to minimising the loss are tiny (Surmenok, 2017). A potential problem with this testing is that optimiser and learning rate are tested separately because the best learning rate will not apply to all optimisers (Mack, 2018). However, the optimiser is tested first so hopefully the results yield the best combination for this parameter.

Set	**Learning Rate Accuracies**						
	1e-7	**1e-6**	**1e-5**	**0.0001**	**0.001**	**0.1**	**1.0**
Training	0.3243	0.5586	0.6133	0.9748	0.9952	0.3333	0.3333
Validation	0.3267	0.5517	0.5733	0.63	0.6283	0.3333	0.3333
Testing	0.3333	0.5667	0.5967	0.59	0.5967	0.3333	0.3333

Table 3. *– CNN model accuracy with different learning rate values, using the ADAM optimizer.*

Results show that for this project, a learning rate of 0.001 provides the best performance with the highest training and testing with 2nd highest validation accuracies. This result is a minimal gain in performance compared to the original learning rate; but the higher rate will train faster and converge reliably. The results from this experiment are to be expected since a learning rate too small can become stuck on un-optimal solutions of high training error, and a learning rate too high can cause the gradient descent to inadvertently increase; instead of decreasing training error (Goodfellow et al, 2017, 429). Hence, the model with learning rates of 1e-7, 0.1, and 1.0 never converge and score ~ 33% accuracy once all the data was exhausted.

Filter Size

Filter or kernel size is a sliding mask that covers all pixels in an image and obtains their dot product to extract a feature from the image (Sahoo, 2018). An odd filter size is most used because with even-sized filters distortions across the layers occur that must be accounted for later, with odd-sized implementation is simpler because the previous layer pixels would be symmetrically around the output – as shown by the following diagram. It is expected that smaller filter sizes lead to better results because the image is scanned with a finer grain and therefore, more features are detected – supported by the results of Camgözlü and Kutlu, (2020). However, a filter size of 1x1 is too small because the features are too finely grained and local; with no information from neighbouring pixels (Pandey, 2020).

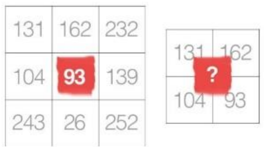


Fig. 14. *– Example of convolution for odd and even sized kernels (Sahoo, 2018).*

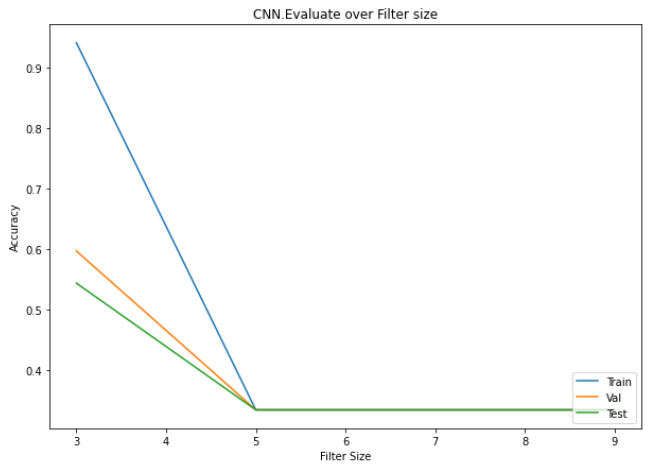


Fig. 15. *– Line graph of CNN model accuracy with varying filter size, x-axis value represents an X by X filter e.g., 5 by 5 or 5x5.*

The results for this experiment were disappointing, all filter sizes other than 3x3 failed to fully converge and stuck at ~ 33.3%. This likely because of the information loss when filtered through in larger amounts i.e., smaller handholds and the circles to label the route could be lost because most of the pixels in the filter might be background, likely leading to misclassification – see Figure 16.

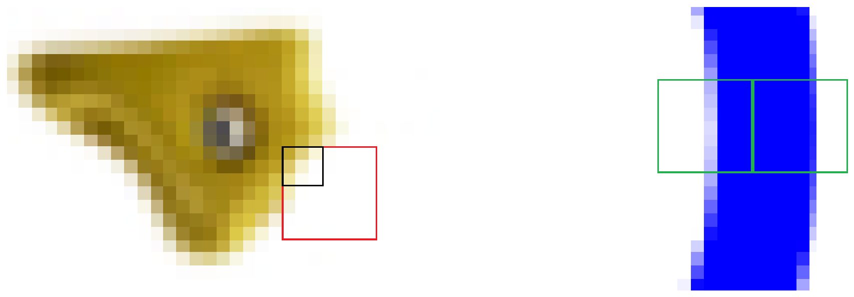


Fig. 16. *– Subsection of a route image to show potential filtering for different sizes, 9x9 filters are in green and red and a 3x3 filter is in black.*

Figure 16 shows an example of the filtering that occurs inside the convolutional layers, on the left it shows what a 3x3 (black) and 9x9 filter (red) could look like; showing that large filters can suffer from non-essential information interference and averaging the edge of a feature to empty. The same could occur when filtering around the circles highlighting the selected route, on the right in green two 9x9 filters could average to a wrong shade of blue and cause confusion for the model. Larger filter sizes perform worse, as shown by the results above and by Ahmed and Karim, (2020), and Han et al, (2018). For the foreseeable testing filter size will remain at 3x3.

Number of Neurons

This section of experimenting is to alter the number of neurons in the hidden layer from the dense network after all the convolutional layers and before the output. Since the CNN only holds one hidden layer, a range of values will be tested to obtain the optimal number of neurons before testing number of hidden layers in the next subsection. It is expected that a lower number of neurons will “lead to under-fitting and high statistical bias” (Krishnan, 2021), but too many neurons will cause over-fitting, high variance, and training time (*ibid)*. With this experiment it is worth keeping in mind that the optimal number of neurons/nodes cannot be calculated because it depends entirely on the application the model is used and the dataset (Brownlee, 2018). Hence, a systematic experimentation to test a range of values is the best approach.

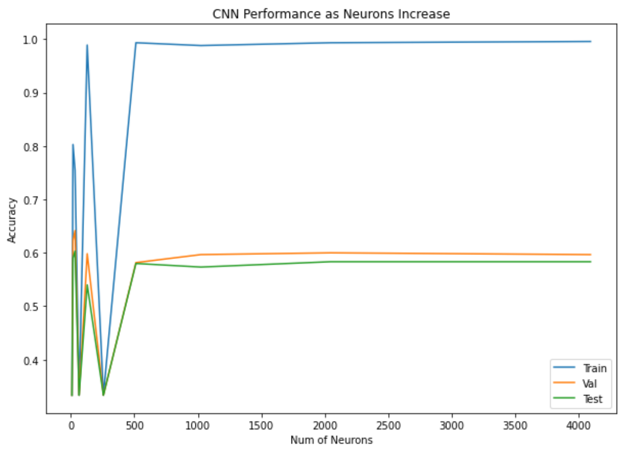


Fig. 17. *– Line graph of CNN model performance with an increasing number of neurons in the hidden layer.*

For this experiment the number of neurons ranges from 32 and doubled up to 4096. From the graph, a lot of noise is visible in the lower values for nodes where the model failed to converge again. Other than an initial spike in performance at 32 neurons, as the amount increases the performance changes are negligible; perhaps with more data the differences between scores would be more distinguishable. It is possible that with a re-run of these failed values, the model would converge. Since the results from this test were inconclusive, to obtain a better picture a further test will be conducted to provide a granular approach to the relationship between number of nodes and performance – see Figure 18.

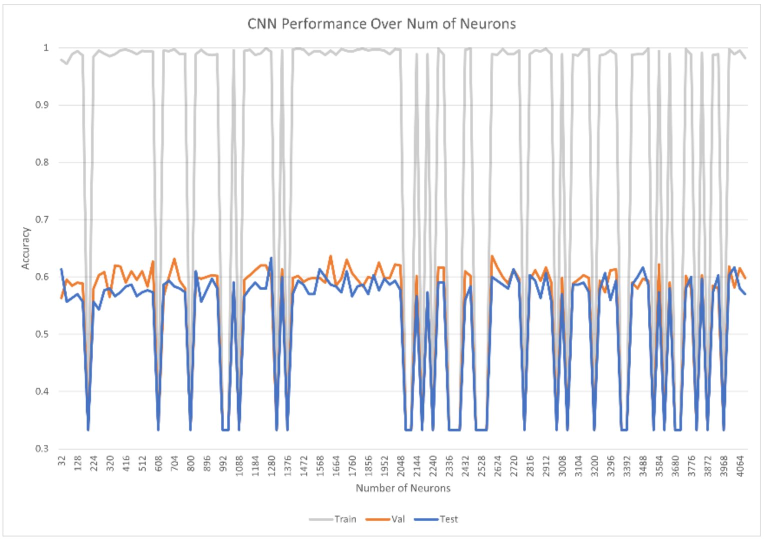


Fig. 18. *– Line graph of the second experiment of increasing number of neurons for CNN performance.*

From the second experiment, more potential values are available, but a lot of noise and failed convergence still occurs. An emerging pattern from these tests is that when the model fails to converge, it fails on all splits of the dataset. Since the training set accuracy is usually high, an altered graph is displayed below to view just the validation and testing sets – see Figure 19.

The testing accuracy peaks again 32 neurons, which shows some performance stability for a lower number of nodes. It peaks highest at 1280 neurons, but this was likely a fluke and would not happen again if re-run. All other points where the accuracy peaks or drops show no pattern as the number of neurons is seemingly random. Although the optimal number of neurons may be one of the peaks in the graph, the amount of noise and instability coupled with the longer time to train, make using 32 as the number of nodes the ideal choice. It is more likely that additional hidden layers will lead to better performance, as shown by the results of Josephine et al, (2021), deeper networks often net better results.

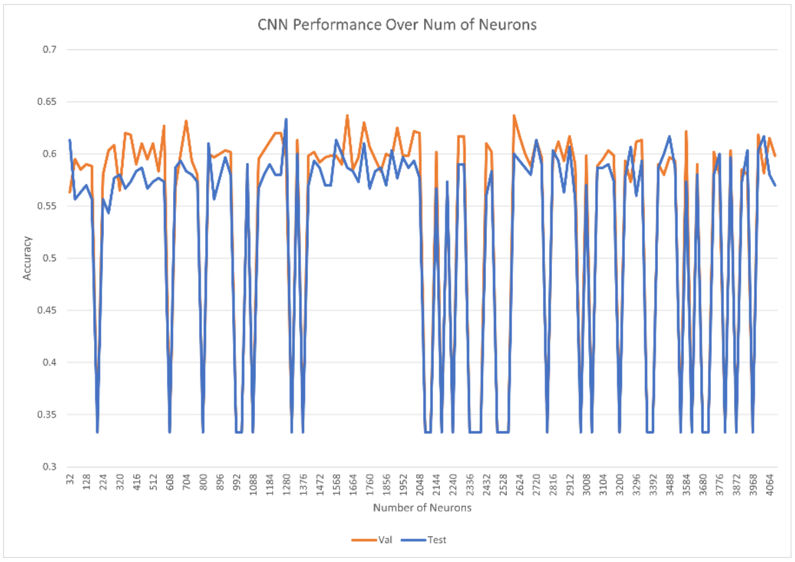
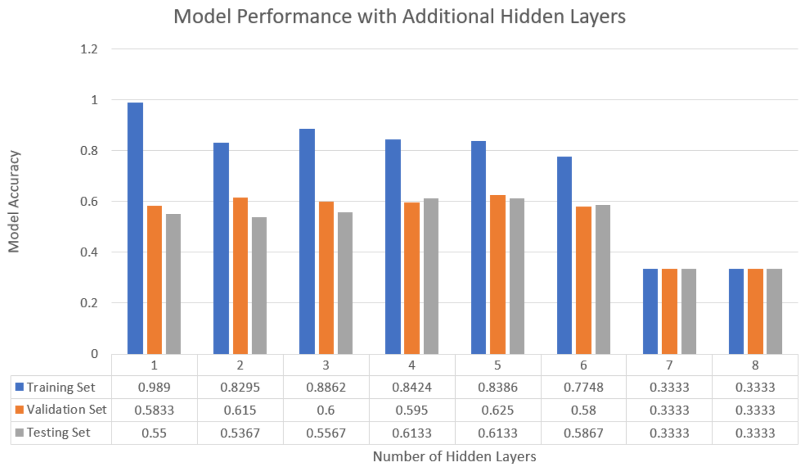


Fig. 19. *– Same graph as Figure 17 but filtering out the training set accuracy.*

Number of Dense Hidden Layers

After testing all the previous hyper-parameters, the final test to make is the effect on performance with extra hidden layers in the densely connected network before the output. For ease of testing the number of neurons in each of the hidden layers is kept constant despite common practice, but this decision should not affect the final performance (Popa. 2017). For this experiment each hidden layer had 32 nodes; obtained from the previous exercise. It is expected that with more layers i.e., more depth to the network, performance should increase as shown by Josephine et al, (2021); and the deep learning network performance on ImageNet (paperswithcode, 2022) – some of which models are implemented in Table 2.



	1	2	3	4	5	6	7	8
■ Training Set	0.989	0.8295	0.8862	0.8424	0.8386	0.7748	0.3333	0.3333
■ Validation Set	0.5833	0.615	0.6	0.595	0.625	0.58	0.3333	0.3333
■ Testing Set	0.55	0.5367	0.5567	0.6133	0.6133	0.5867	0.3333	0.3333

Fig. 20. *– Bar chart of the model accuracy with additional hidden layers in the dense network.*

Results show a steady increase in testing performance with more hidden layers up to four or five where the accuracy stagnates and fails to converge above 6. This matches the results of Uzair and Jamil, (2020), and Josephine et al, (2021), with the MoonBoard dataset breaks this trend, more depth does not always lead to better performance. Too many hidden layers increase time complexity, over-complicates model, and will take longer to converge. This could be due to the number of neurons, Shafi et al, (2006) find success in using two hidden layers with high numbers of nodes, for this model it was tested that 32 nodes were optimal but with additional layers more might be needed. Additionally, with additional layers, more epochs might be needed to accommodate more weight changes in training and prevent under-fitting (Sharma, 2017). The optimal number of hidden layers for this dataset is 4, this result is supported by the findings of Panchal and Panchal, (2014).

4.4.2 Summary of Testing

To conclude, the testing and optimisation of the CNN resulted in an approximate 0.06 increase in performance and some additional stability, which given the extensive testing and time spent is a disappointing result. However, this performance is now comparable to the pre-trained models tested in Table 2 and is achieved with a simpler model architecture, smaller time complexity and use of resources. The final model performance is displayed by the following, again these values are not consistently reproducible due to random seeding – as brought up in Section 4.4.

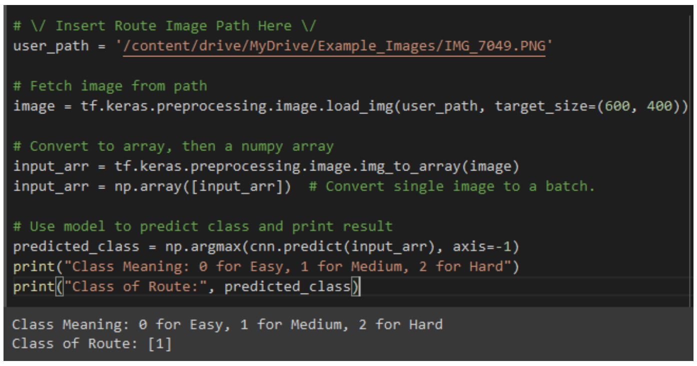
The potential further improvements and downfalls of this testing will be discussed later in Section 6.

Model	**Model Accuracy on each Dataset**								
	Train	**Val**	**Test**	**GS Train**	**GS Val**	**GS Test**	**No BG Train**	**No BG Val**	**No BG Test**
Original CNN	0.989	0.5833	0.55	0.7071	0.615	0.6067	0.969	0.5783	0.6067
Optimised CNN	0.8424	0.595	0.6133	0.843	0.59	0.5988	0.929	0.575	0.6367

Table 4. *– Model performance comparison for original implementation and the optimised version.*

4.5 Future Operation

To simulate how this artefact would be used in a real-world scenario, this section will detail how a user would utilise this software to find the grade boundary of a MoonBoard climbing route. With the fully trained model a climber can use a picture of a route either from the MoonBoard application or perhaps with their phone camera, and this project artefact will predict the grade boundary of the new route as either ‘Easy’, ‘Medium’, or ‘Hard’. Unfortunately, because the artefact is not packages into an application, the user would have to load their image into the code and run it to receive an output prediction of its class. See Figure 20 for an example output from prediction.



```
# \/ Insert Route Image Path Here \/
user_path = '/content/drive/MyDrive/Example_Images/IMG_7049.PNG'

# Fetch image from path
image = tf.keras.preprocessing.image.load_img(user_path, target_size=(600, 400))

# Convert to array, then a numpy array
input_arr = tf.keras.preprocessing.image.img_to_array(image)
input_arr = np.array([input_arr])  # Convert single image to a batch.

# Use model to predict class and print result
predicted_class = np.argmax(cnn.predict(input_arr), axis=-1)
print("Class Meaning: 0 for Easy, 1 for Medium, 2 for Hard")
print("Class of Route:", predicted_class)
```

```
Class Meaning: 0 for Easy, 1 for Medium, 2 for Hard
Class of Route: [1]
```

Fig. 21. *– Snippet of the artefact a user would use to predict the class of an input image, with an example classification output.*

With this block the user can pass any MoonBoard route image and receive a classification. By changing the file path, the image will automatically be resized and converted to have its class predicted by the CNN with an approximate 60% accuracy.

Chapter 5

5 Project Conclusion

In this project the question is raised to whether indoor climbing routes can be classified by difficulty; to attempt resolving the disadvantages of manual grading outlined in the introduction. By analysing the model score of several pre-trained models and a new developed CNN, the research question was answered with an average accuracy of 60%, With this artefact; any new image of a MoonBoard route can be classified into three difficulty ratings (Easy, Medium, or Hard).

The methodology to answer this question, was to apply image classification to raw MoonBoard images with convolutional neural networks. This approach was used because of the effectiveness and popularity of CNNs or image classification. The limitations to this research stems from the dataset and the similarity between classes. From the methodology and evaluation, the question is raised to whether a layer of feature extraction would be beneficial for the classification; like the work by Duh and Chang, (2021).

Referring to the problem statement and literature review, this project manages to reliably classify MoonBoard routes solely with just an image. No amount of pre-processing or calculations are required from a user to obtain a difficulty prediction. This project contributes to automatic climbing route classification, that could eventually be part of a full application to assist route setting and the grade assigned.

Based on these conclusions, practitioners should consider the following section any for future development of this project.

5.1 Further Work

This section discusses the ways in which this project could be expanded given additional time, planning and resources – which will act as recommendations for other practitioners. It will cover some discussion of the relevant literature and how some stretch objectives could be attempted. Other stretch objectives not discussed here; will be in Section 6.

Comparing Literature to this Project

Some of the methodologies in the discussed literature is potentially transferable to the MoonBoard problem space and show possible avenues for this project to continue with further development.

One of the stretch objectives of this project was to generate new MoonBoard climbing routes. Philips et al, (2011), Geary and Valdez, (2019), Stapel, (2020), and Lo, (2020) present various methods to complete that could be applicable to this artefact. The most apt method is by Geary and Valdez, (2019), their use of generative adversarial networks would complement this project well with additional expansion, by providing an experience on top of the classification so users can generate routes at their desired grade. Additionally, Dobles et al, (2017); and Tai et al, (2020) both mention the merits of these networks in this problem space.

With further development, a method of classification using the Whole-History-Rating (WHR) presented by Scarff, (2020) could be effective for classifying MoonBoard routes. Since their dataset and the MoonBoard dataset are similar where both consist of information on climbing routes about the grading by user input with community rating, the WHR method could simply be applied to MoonBoards.

Chapter 6

6 Reflective Analysis

Given the hindsight to analyse and reflect on the work completed in this project, the things that went well were the analysis of the problem space or dataset, and overall time management. The downfalls in this project were the lack of software development / project management strategies, errors I made during development, missing optimisation methods, and shortage of performance metrics. In this section the reasoning why the stretch objectives were not attempted are also explained.

6.1 Effective Areas of the Project

The first positive reflection I made was my analysis of the problem space and dataset. I completed a critical discussion of the MoonBoard and the dataset which built the justification and importance of this project. Other climbers in the community could benefit from this discussion as well as other projects in this field. The second area that went well in this project was the time management, all planned tasks were complete within the given time with the space to compare many pre-trained models, and attempt lots of optimisation options.

6.2 Downfalls of the Project

The first downfall of this project was the lack of software development / project management strategies. Although the planned tasks were completed in-time, the use of a Kanban would provide better visualisation of the tasks to complete and lead to better communication with my supervisor (Siderova, 2018). With a Kanban board we could collaborate more efficiently, and they would be able to better track my progress.

A second downfall was during development I made several avoidable errors e.g., the No BG dataset was mismanaged leading to a few days of training wasted on incorrect data. To avoid these types of mistakes in future development, I would use more automated approaches to eliminate the risk of human error.

Another downfall of this project is the optimisation methods implemented. While Section 4.4.1 does cover a range of optimisation methods by tuning the hyper-parameters; but fails to cover some methods that could net performance gains. Firstly, regularisation techniques like dropout or L1/L2 are not tested, effects of these are shown by Srivastava et al, (2014) and Patel et al, (2022). Similarly, decaying learning rates are not tested despite their common use (You et al, 2019); however, Smith et al, (2017) propose increasing batch sizes in training leads to better performance. All these parameters will have significant effects on model performance but were not tested in this project due to time constraints. Moreover, as seen by the project results; parameter fine-tuning is not the best strategy for boosting performance (Becherer et al, 2019), it is more likely performance gains will come from the dataset – as mentioned in Section 4.2.1. With further development a full study of every parameter and their effect on performance can be tested; perhaps using an auto-tuning tool like KerasTuner.

A final downfall of this project was the shortage of performance metrics. This project only takes into the account the model accuracy, loss and confusion matrixes, other metrics like F1 score, recall & precision, and AUC would provide additional statistics and indicators of the models' performance and stability (Bose, 2020). This metrics were neglected or unconsidered until it was too late, with more time these additional measurements would provide better analytics to the model performance.

6.3 Stretch Objectives

Unfortunately, the stretch objectives were not attempted because the time to train the model in optimisation was unknown. Therefore, I ran out of time to complete any stretch objectives planned out in the introduction. With no constraints the stretch objectives could be completed, starting by following the mentioned literature in Section 5.1 and researching further with additional resources and outside help.

6.4 Summary of Reflective Analysis

To conclude my reflective analysis of this project, the task to classify MoonBoard route images into grade boundaries was successful. It was able to achieve and average accuracy of 60% for predicting the difficulty rating of any new route. Compared to similar literature, it performs comparably in-terms of accuracy; however, their projects have more classes. Given more time, the project could extend to include the missed techniques and resolve the downfalls mentioned in this section.

7 References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M. and Ghemawat, S., (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*. Available from: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf [accessed 14th April 2022].
2. Ahmed, T. (2018). *Would you recommend using existing libraries or frameworks or building it from scratch in general? And why?* [Discussion]. Available from: https://www.quora.com/Would-you-recommend-using-existing-libraries-or-frameworks-or-building-it-from-scratch-in-general-And-why [accessed 12th April 2022].
3. Ahmed, W.S., Karim, A.A., (2020). *The impact of filter size and number of filters on classification accuracy in CNN*. In 2020 International conference on computer science and software engineering (CSASE) (pp. 88-93). IEEE. Available from: https://ieeexplore.ieee.org/document/9142089 [accessed 28th April 2022].
4. Al Zorgani, M. and Ugail, H., (2018). *Comparative study of image classification using machine learning algorithms*. Technical report. In: The 2nd Annual Innovative Engineering Research Conference (AIERC 2018), Bradford, UK, October. 17, 2018. Unknown. Available from https://easychair.org/publications/preprint_download/bRtJ [accessed 12th May 2022].
5. Bajpai, S. (2020). *Difference between Scikit-Learn and TensorFlow*. United States: medium.com. Available from: https://medium.com/@shvbajpai/difference-between-scikit-learn-and-tensorflow-b6ad2f7b840c [accessed 11th April 2022].
6. Becherer, N., Pecarina, J., Nykl, S. and Hopkinson, K., (2019). *Improving optimization of convolutional neural networks through parameter fine-tuning*. Neural Computing and Applications, 31(8), pp.3469-3479. Available from: https://link.springer.com/article/10.1007/s00521-017-3285-0 [accessed 23rd May 2022].
7. Bennett, M. and Min, Y., (2020). *Causal Explanations of Image Misclassifications*. arXiv preprint arXiv:2006.15739. Available from: https://arxiv.org/pdf/2006.15739.pdf [accessed 22nd May 2022].

8. Bera, S. and Shrivastava, V.K., (2020). *Analysis of various optimizers on deep convolutional neural network model in the application of hyperspectral remote sensing image classification*. International Journal of Remote Sensing, 41(7), pp.2664-2683. Available from: https://www.tandfonline.com/doi/full/10.1080/01431161.2019.1694725 [accessed 26th April 2022].
9. Bezanson, J., Edelman, A., Karpinski, S. and Shah, V.B., (2017). *Julia: A fresh approach to numerical computing.* SIAM review, 59(1), pp.65-98. Available from: https://epubs.siam.org/doi/10.1137/141000671 [accessed 14th April 2022].
10. Boesch, G. (2022). *A Complete Guide to Image Classification in 2022.* Switzerland: viso.ai. Available at: https://viso.ai/computer-vision/image-classification/ [accessed 12th May 2022].
11. Bose, B. (2020). *Performance Metrics: Classification Model.* Unknown: Analytics Vidhya. Available from: https://medium.com/analytics-vidhya/performance-metrics-classification-model-69a5546b118c [accessed 23rd May 2022].
12. Bradski, G., (2000). *The OpenCV library*. Dr. Dobb's Journal: Software Tools for the Professional Programmer, 25(11), pp.120-123. Available from: https://elibrary.ru/item.asp?id=4934581 [accessed 14th April 2022].
13. Brownlee, J. (2016). *How to Improve Deep Learning Performance.* Unknown: machinelearningmastery.com. Available from: https://machinelearningmastery.com/improve-deep-learning-performance/ [accessed 21st April 2022].
14. Brownlee, J. (2018). *How to Configure the Number of Layers and Nodes in a Neural Network.* Unknown: machinelearningmastery.com. Available from: https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/ [accessed 28th April 2022].
15. De Bruijn, B. (2019). *Let's Talk About Grades.* Canada: gripped.com. Available from: https://gripped.com/indoor-climbing/lets-talk-about-grades/ [accessed 22nd May 2022].
16. Camgözlü, Y. and Kutlu, Y., (2020). *Analysis of Filter Size Effect in Deep Learning*. arXiv preprint arXiv:2101.01115. Available from: https://arxiv.org/ftp/arxiv/papers/2101/2101.01115.pdf [accessed 25th April 2022].

17. Chadwick, R. (2022). *Gantt Charts! Plan for success.* Unknown: ryanstutorials.net. Available from: https://ryanstutorials.net/software-design-and-development/gantt-charts.php [accessed 3rd May 2022].
18. Chandana. (2022). *Scrum Project Management: Advantages and Disadvantages.* India: simplilearn.com. Web link: https://www.simplilearn.com/scrum-project-management-article [accessed 8th May 2022].
19. Chollet, F., (2017). *Xception: Deep learning with depthwise separable convolutions*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1251-1258). Available from: https://arxiv.org/pdf/1610.02357.pdf [accessed 22nd May 2022].
20. ChssPly76. (2017). *How do you decide whether to use a library or write your own implementation?* [Discussion]. Web link: https://stackoverflow.com/q/1236256 [accessed 12th April 2022].
21. Cowpig. (2018). *When do you use a library vs. implement something yourself?* [Discussion]. Available at: https://news.ycombinator.com/item?id=16882140 [accessed 12th April 2022].
22. Cyb70289. (2017). *Why is Keras way slower than Sklearn?* [Discussion]. Web link: https://stackoverflow.com/q/42732881 [accessed 11th April 2022].
23. Dhaduk, H. (2021). *Why White Box Testing is an Essential Part of Software Testing*. United States: hackernoon.com. Available from: https://hackernoon.com/why-white-box-testing-is-an-essential-part-of-software-testing [accessed 16th May 2022].
24. Dialani, P. (2021). *Why should Python be used in Machine Learning.* Greece: analyticsinsight.net. Web link: https://www.analyticsinsight.net/why-should-python-be-used-in-machine-learning/ [accessed 13th April 2022].
25. Diebold, P. and Mayer, U., (2017). On the usage and benefits of agile methods & practices. In International Conference on Agile Software Development (pp. 243-250). Springer, Cham. Available from: https://library.oapen.org/bitstream/handle/20.500.12657/27861/1002143.pdf?sequence=1#page=251 [accessed 7th May 2022].
26. Dobles, A., Sarmiento, J.C. and Satterthwaite, P., (2017). *Machine learning methods for climbing route classification*. Web link: http://cs229.stanford.edu/proj2017/final-reports/5232206.pdf [accessed 3rd October 2021].

27. Draper, N., (2016). *14 Climbing grades*. The Science of Climbing and Mountaineering, p.227. Available from: http://dx.doi.org/10.4324/9781315682433 [accessed 22nd February 2022].

28. Duh, Y.S. and Chang, R., (2021). *Recurrent Neural Network for MoonBoard Climbing Route Classification and Generation*. ArXiv preprint arXiv: 2102.01788. Available at: https://arxiv.org/abs/2102.01788 [accessed 3rd October 2021].

29. Ebert, A., Schmid, K., Marouane, C. and Linnhoff-Popien, C., (2017). *Automated recognition and difficulty assessment of boulder routes.* In International Conference on IoT Technologies for HealthCare (pp. 62-68). Springer, Cham. Web link: https://link.springer.com/chapter/10.1007/978-3-319-76213-5_9 [accessed 10th January 2022].

30. Elliot, T. (2019). *The State of the Octoverse: machine learning.* United States: github.blog. Available at: https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/ [accessed 13th April 2022].

31. Forbes Technology Council. (2016). *The Benefits of Using Agile Software Development.* United States: forbes.com. Available from: https://www.forbes.com/sites/forbestechcouncil/2016/05/09/the-benefits-of-using-agile-software-development/?sh=50d58178b0f8 [accessed 7th May 2022].

32. Gao, K., Mei, G., Piccialli, F., Cuomo, S., Tu, J. and Huo, Z., (2020). *Julia language in machine learning: Algorithms, applications, and open issues.* Computer Science Review, 37, p.100254. Available from: https://www.sciencedirect.com/science/article/pii/S157401372030071X [accessed 14th April 2022].

33. Geary, C. and Valdez, J., (2019). *Climb a-GAN: Generation of Rock-Climbing Problems* [pre-print]. In Proceedings of ACM Conference (Conference’17). USA, 6 pages. Available from: https://cpsc459-bim.gitlab.io/f19/assets/reports/climbing.pdf [accessed 10th January 2022].

34. Gilmore, N. (2016). *The Advantages of Gantt Charts in Project Management*. United States: teamgantt.com. Available from: https://www.teamgantt.com/blog/10-benefits-of-using-a-gantt-chart-of-your-next-project [accessed 3rd Mary 2022].

35. Goodfellow, I., Bengio, Y. and Courville, A., 2017. Deep learning (adaptive computation and machine learning series). *Cambridge*

Massachusetts, pp.321-359. Available from: https://www.academia.edu/38223830/Adaptive_Computation_and_Machine_Learning_series_Deep_learning_The_MIT_Press_2016_pdf [accessed 27th April 2022].

36. Google. (2017). *Google Colaboratory – Google Research.* [Software]. United States. Available from: https://colab.research.google.com/?utm_source=scs-index [accessed 1st April 2022].
37. Gupta, S. (2021). *TensorFlow vs Sciki-Learn: How Do They Compare?* United States: springboard.com. Available from: https://www.springboard.com/blog/data-science/scikit-learn-vs-tensorflow/ [accessed 11th April 2022].
38. Hasabo, I. (2020). *Image Classification using Machine Learning and Deep Learning.* United States: medium.com. Available from [12th May 2022].
39. Han, S., Meng, Z., Li, Z., O'Reilly, J., Cai, J., Wang, X. and Tong, Y., (2018). *Optimizing filter size in convolutional neural networks for facial action unit recognition*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 5070-5078). Available from: https://openaccess.thecvf.com/content_cvpr_2018/papers/Han_Optimizing_Filter_Size_CVPR_2018_paper.pdf [accessed 28th April 2022].
40. Han, X., Zhang, Z., Ding, N., Gu, Y., Liu, X., Huo, Y., Qiu, J., Yao, Y., Zhang, A., Zhang, L. and Han, W., (2021). *Pre-trained models: Past, present and future*. AI Open, 2, pp.225-250. Web link: https://www.sciencedirect.com/science/article/pii/S2666651021000231 [accessed 15th May 2022].
41. Harris, C.R., Millman, K.J., Van Der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J. and Kern, R., (2020). *Array programming with NumPy*. Nature, 585(7825), pp.357-362. Web link: https://www.nature.com/articles/s41586-020-2649-2 [accessed 14th April 2022].
42. He, K., Girshick, R. and Dollár, P., (2019). *Rethinking imagenet pre-training*. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 4918-4927). Web link: https://arxiv.org/pdf/1811.08883.pdf [accessed 15th May 2022].
43. He, K., Zhang, X., Ren, S. and Sun, J., (2016). *Deep residual learning for image recognition*. In Proceedings of the IEEE conference on

computer vision and pattern recognition (pp. 770-778). Web link: https://arxiv.org/abs/1512.03385 [accessed 30th March 20th 2022].

44. Hendrycks, D., Lee, K. and Mazeika, M., (2019). *Using pre-training can improve model robustness and uncertainty*. In International Conference on Machine Learning (pp. 2712-2721). PMLR. Available from: https://proceedings.mlr.press/v97/hendrycks19a/hendrycks19a.pdf [accessed 15th May 2022].
45. Houghton, A., Mann, R. and Herbert, J., (2020). *Moon Board Climbing.* GitHub repository. Available from: https://github.com/andrew-houghton/moon-board-climbing [accessed 22nd February 2022].
46. Huang, G., Liu, Z., Van Der Maaten, L. and Weinberger, K.Q., (2017). *Densely connected convolutional networks*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708). Available from: https://arxiv.org/pdf/1608.06993.pdf [accessed 22nd May 2022].
47. Hunter, J.D., (2007). *Matplotlib: A 2D graphics environment*. Computing in science & engineering, 9(03), pp.90-95. Web link: https://ieeexplore.ieee.org/document/4160265 [accessed 14th April 2022].
48. Innes, M., Karpinski, S., Shah, V., Barber, D., Saito Stenetorp, P.L.E.P.S., Besard, T., Bradbury, J., Churavy, V., Danisch, S., Edelman, A. and Malmaud, J., (2018). *On machine learning and programming languages*. Association for Computing Machinery (ACM). Available from: https://discovery.ucl.ac.uk/id/eprint/10051391/1/37.pdf [accessed 14th April 2022].
49. Josephine, V.H., Nirmala, A.P. and Alluri, V.L., (2021). *Impact of Hidden Dense Layers in Convolutional Neural Network to enhance Performance of Classification Model*. In IOP Conference Series: Materials Science and Engineering (Vol. 1131, No. 1, p. 012007). IOP Publishing. Available from: https://iopscience.iop.org/article/10.1088/1757-899X/1131/1/012007/meta [accessed 28th April 2022].
50. Kandel, I. and Castelli, M., (2020). *The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset*. ICT express, 6(4), pp.312-315. Available from: https://www.sciencedirect.com/science/article/pii/S2405959519303455 [accessed 25th April 2022].

51. Kempen, L., (2018). *A fair grade: assessing difficulty of climbing routes through machine learning*. Formal Methods and Tools, University of Twente. Available at: https://fmt.ewi.utwente.nl/media/30-TScIT_paper_46.pdf [accessed 3rd October 2021].
52. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M. and Tang, P.T.P., (2016). On large-batch training for deep learning: Generalization gap and sharp minima. arXiv pre-print arXiv:1609.04836. Available from: https://arxiv.org/pdf/1609.04836.pdf [accessed 25th April 2022].
53. Koert, B. (2018). *Use of existing frameworks and libraries. Why?* [Discussion]. Would you recommend using existing libraries or frameworks or building it from scratch in general? And why? [Online]. Available from: https://qr.ae/pv2rUl [accessed 12th April 2022].
54. Kosmalla, F., Daiber, F. and Krüger, A., (2015). Climbsense: Automatic climbing route recognition using wrist-worn inertia measurement units. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (pp. 2033-2042). Web link: https://dl.acm.org/doi/pdf/10.1145/2702123.2702311 [accessed 10th January 2022].
55. Krishnan, S. (2021). *How to determine the number of layers and neurons in the hidden layer?* United States: medium.com. Available from: https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3 [accessed 28th April 2022].
56. Krizhevsky, A., Sutskever, I. and Hinton, G.E., (2012). *Imagenet classification with deep convolutional neural networks*. Advances in neural information processing systems, 25. Web link: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html [accessed 22nd February 2022].
57. Kumar, S. (2021). *Why Python is best for AI, ML, and Deep Learning.* United States: rtinsights.com. Available from: https://www.rtinsights.com/why-python-is-best-for-ai-ml-and-deep-learning/ [accessed 13th April 2022].
58. Ladha, C., Hammerla, N.Y., Olivier, P. and Plötz, T., (2013). *ClimbAX: skill assessment for climbing enthusiasts*. In Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing (pp. 235-244). Available from:

https://dl.acm.org/doi/pdf/10.1145/2493432.2493492 [accessed 22nd February 2022].

59. Landup, D. (2022). *Split Train, Test and Validation Sets with TensorFlow Datasets.* United States: StackAbuse.com. Available at: https://stackabuse.com/split-train-test-and-validation-sets-with-tensorflow-datasets-tfds/ [accessed 14th April 2022].
60. Lau, Y., Sim, W., Chew, K., Ng, Y., and Salam, Z.A.A., (2021). *Understanding how noise affects the accuracy of CNN image classification.* Journal of Applied Technology and Innovation (e-ISSN: 2600-7304), 5(2), p.23. Available from: https://dif7uuh3zqcps.cloudfront.net/wp-content/uploads/sites/11/2021/03/15102550/Volume5_Issue2_Paper4_2021.pdf [accessed 14th April 2022].
61. Layton, M. (2016). *10 Key Benefits of Scrum.* United States: dummies.com. Available from: https://www.dummies.com/article/business-careers-money/business/project-management/10-key-benefits-of-scrum-143117/ [accessed 8th May 2022].
62. Lo, K.H., (2020). *Embedding and generation of indoor climbing routes with variational autoencoder*. ArXiv preprint arXiv: 2009.13271. Web link: https://arxiv.org/pdf/2009.13271.pdf [accessed 3rd October 2021].
63. Lord, A. (2021). *Trial and Error: Fail and Error: Fail your way to Success.* Unknown: linkedin.com. Available from: https://www.linkedin.com/pulse/trial-error-method-success-andre-lord [accessed 16th May 2022].
64. Luashchuk, A. (2019). *Why I Think Python is perfect for Machine Learning and Artificial Intelligence.* United States: towardsdatascience.com. Available at: https://towardsdatascience.com/8-reasons-why-python-is-good-for-artificial-intelligence-and-machine-learning-4a23f6bed2e6 [accessed 13th April 2022].
65. Luke, J.J., Joseph, R. and Balaji, M., (2019*). Impact of image size on accuracy and generalization of convolutional neural networks*. IJRAR February 2019, Volume 6, Issue 1, 70-80. Available from: https://www.researchgate.net/publication/332241609_IMPACT_OF_IMAGE_SIZE_ON_ACCURACY_AND_GENERALIZATION_OF_CONVOLUTIONAL_NEURAL_NETWORKS [accessed 14th April 2022].
66. Luo, C., Li, X., Wang, L., He, J., Li, D. and Zhou, J., (2018). *How does the data set affect cnn-based image classification perfor-*

mance?. In 2018 5th International conference on systems and informatics (ICSAI) (pp. 361-366). IEEE. Available from: https://ieeexplore.ieee.org/document/8599448 [accessed 23rd May 2022].

67. Mack, D. (2018). *How to pick the best learning rate for your machine learning project.* United States: medium.com. Available from: https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2 [accessed 25h April 2022].

68. Maruyama, T., Hayashi, N., Sato, Y., Hyuga, S., Wakayama, Y., Watanabe, H., Ogura, A. and Ogura, T., (2018). *Comparison of medical image classification accuracy among three machine learning methods*. Journal of X-ray Science and Technology, 26(6), pp.885-893. Available from: https://pubmed.ncbi.nlm.nih.gov/30223423/ [accessed 12th May 2022].

69. Masters, D. and Luschi, C., (2018). *Revisiting small batch training for deep neural networks*. ArXiv preprint arXiv: 1804.07612. Available from: https://arxiv.org/abs/1804.07612 [accessed 25th April 2022].

70. McKinney, W., (2010). *Data structures for statistical computing in python*. In Proceedings of the 9th Python in Science Conference (Vol. 445, No. 1, pp. 51-56). Available from: https://conference.scipy.org/proceedings/scipy2010/pdfs/mckinney.pdf [accessed 14th April 2022].

71. Mikhalchuk, O. (2020). *The importance of unit testing, or how bugs found in time will save you money.* Unknown: fortegrp.com. Available at: https://fortegrp.com/the-importance-of-unit-testing/ [accessed 16th May 2022].

72. Mishra, P. (2019). *Why are Convolutional Neural Networks good for image classification.* United States: medium.com. Available from: https://medium.datadriveninvestor.com/why-are-convolutional-neural-networks-good-for-image-classification-146ec6e865e8 [accessed 12th May 2022].

73. Moon Climbing, (2019). *Moon: MoonBoard.* United Kingdom: moonboard.com. Available from: https://www.moonboard.com/ [accessed 22nd February].

74. Nikola, O. (2021). *Python Library vs Implementation from Scratch: 7 Things to Consider*. United States: hackernoon.com. Available from: https://hackernoon.com/python-library-vs-implementation-from-scratch-7-things-to-consider [accessed 12th April 2022].

75. Panchal, F.S. and Panchal, M., (2014). *Review on methods of selecting number of hidden nodes in artificial neural network.* International Journal of Computer Science and Mobile Computing, 3(11), pp.455-464. Available from: https://www.ijcsmc.com/docs/papers/November2014/V3I11201499a19.pdf [accessed 22nd May 2022].
76. Pandey, S. (2020). *How to choose the size of the convolution filter or Kernel size for CNN?* United States: medium.com. Available at: https://medium.com/analytics-vidhya/how-to-choose-the-size-of-the-convolution-filter-or-kernel-size-for-cnn-86a55a1e2d15 [accessed 25th April 2022].
77. Paperswithcode Team. (2022). *Image Classification on ImageNet* .United States: paperswithcode.com. Web link: https://paperswithcode.com/sota/image-classification-on-imagenet [accessed 12th May 2022].
78. Patel, P. (2018). *Why Python is the most popular language used for Machine Learning.* United States: medium.com. Web link: https://medium.com/@UdacityINDIA/why-use-python-for-machine-learning-e4b0b4457a77 [accessed 13th April 2022].
79. Patel, V., Shukla, S., Shrivastava, S. and Gyanchandani, M., (2022). *Regularized CNN for Traffic Sign Recognition.* In 2022 International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN) (pp. 1-5). IEEE. Available from: https://ieeexplore.ieee.org/document/9761341 [accessed 23rd May 2022].
80. Paterska, P. (2021). *Agile revolutionized modern project management and software development. What are some of the agile benefits?* Poland: elpassion.com. Available from: https://www.elpassion.com/blog/10-benefits-of-agile [accessed 7th May 2022].
81. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., (2011). *Scikit-learn: Machine learning in Python*. The Journal of machine Learning research, 12, pp.2825-2830. Available at: https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html [accessed 14th April 2022].
82. Perfecto Team. (2022). *Different Types of Testing in Software.* Unknown: perfecto.io. Available from: https://www.perfecto.io/resources/types-of-testing [accessed 13th May 2022].
83. Pfeil, J., Mitani, J. and Igarashi, T., (2011). *Interactive climbing route design using a simulated virtual climber. In SIGGRAPH Asia 2011*

Sketches (pp. 1-2). Available from: https://dl.acm.org/doi/pdf/10.1145/2077378.2077381 [accessed 10th January 2022].

84. Phillips, C., Becker, L. and Bradley, E., (2012). *Strange beta: An assistance system for indoor rock-climbing route setting.* Chaos: An Interdisciplinary Journal of Nonlinear Science, 22(1), p.013130. [accessed 3rd October 2021].
85. Popa, R. (2017). *Why is it common in Neural Network to have a decreasing number of neurons as the Network becomes deeper?* [Discussion]. Available from: https://qr.ae/pvAc8M [accessed 18th April 2022].
86. Pplonski. (2020). *what's the difference between Scikit-Learn and TensorFlow? Is it possible to use them together?* [Discussion]. Web link: https://stackoverflow.com/a/64156418 [accessed 11th April 2022].
87. Puget, J.F. (2017). *The Most Popular Language for Machine Learning.* United States: Inside Machine Learning. Available from: https://medium.com/inside-machine-learning/the-most-popular-language-for-machine-learning-is-46e2084e851b [accessed 13th April 2022].
88. Richter, M.L., Byttner, W., Krumnack, U., Wiedenroth, A., Schallner, L. and Shenk, J., (2021). *(Input) Size Matters for CNN Classifiers*. In International Conference on Artificial Neural Networks (pp. 133-144). Springer, Cham. Available from: https://arxiv.org/pdf/2102.01582.pdf [accessed 10th May 2022].
89. Robinson, D. (2017). *The Incredible Growth of Python.* United States: stackoverflow.blog. Available from: https://stackoverflow.blog/2017/09/06/incredible-growth-python/ [accessed 13th April 2022].
90. Russell, S. (2021). *Common Route Setting Injuries and How to Prevent Them.* United States: unionpt.com. Available from: https://www.unionpt.com/route-setting-injuries/ [accessed 22nd May 2022].
91. Sahoo, S. (2018). *Deciding optimal kernel size for CNN.* United States: towardsdatascience.com. Available from: https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363 [accessed 25th April 2022].
92. Sarkar, T., (2019). *A single function to streamline image classification with Keras*. United States: towardsdatascience.com. Available

from: https://towardsdatascience.com/a-single-function-to-streamline-image-classification-with-keras-bd04f5cfe6df [accessed 10th January 2022].

93. Saragih, A. (2019). *4 Methods to Boost the Accuracy of a Neural Network Model.* United States: medium.com. Available from: https://medium.com/@amrianto.saragih/4-methods-to-boost-the-accuracy-of-a-neural-network-model-bb694e650a69 [accessed 14th April 2022].
94. Scarff, D., (2020). *Estimation of Climbing Route Difficulty using Whole-History Rating*. ArXiv preprint arXiv: 2001.05388. Available at: https://arxiv.org/pdf/2001.05388.pdf [accessed 10th January 2022]. *Moon: MoonBoard.* London: moonboard.com. Available from: https://www.moonboard.com/ [accessed 22nd February 2022].
95. Schmidt, R.M., Schneider, F. and Hennig, P., (2021). *Descending through a crowded valley-benchmarking deep learning optimizers*. In International Conference on Machine Learning (pp. 9367-9376). PMLR. Available from: http://proceedings.mlr.press/v139/schmidt21a/schmidt21a.pdf [accessed 16th May 2022].
96. Shafi, I., Ahmad, J., Shah, S.I. and Kashif, F.M., (2006). *Impact of varying neurons and hidden layers in neural network architecture for a time frequency application*. In 2006 IEEE International Multitopic Conference (pp. 188-193). IEEE. Available from: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4196403 [accessed 22nd May 2022].
97. Shao, C. (2019). *Approach pre-trained deep learning models with caution.* United States: medium.com. Available at: https://medium.com/comet-ml/approach-pre-trained-deep-learning-models-with-caution-9f0ff739010c [accessed 15th May 2022].
98. Shao, C. (2019). *Properly Setting the Random Seed in ML Experiments. Not as Simple as You Might Imagine.* Unknown: opendatascience.com. Available from: https://opendatascience.com/properly-setting-the-random-seed-in-ml-experiments-not-as-simple-as-you-might-imagine/ [accessed 19th May 2022].
99. Sharma, P. (2020). *7 Popular Image Classification Models in ImageNet Challenge Competition History*. Unknown: machinelearningknowledge.ai. Available from: https://machinelearningknowledge.ai/popular-image-classification-models-in-imagenet-challenge-ilsvrc-competition-history/ [accessed 12th May 2022].

100. Sharma, S. (2017). *Epoch vs Batch Size vs Iterations.* United States: towardsdatascience.com. Available from: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9 [accessed 25th April 2022].
101. Shi, S., Wang, Q., Xu, P. and Chu, X., (2016). *Benchmarking state-of-the-art deep learning software tools.* In 2016 7th International Conference on Cloud Computing and Big Data (CCBD) (pp. 99-104). IEEE. Web link: https://arxiv.org/pdf/1608.07249.pdf [accessed 11th April 2022].
102. Shubhamsingh10. (2021). *Difference between Compiled and Interpreted Language.* India: geeksforgeeks.org. Available from: https://www.geeksforgeeks.org/difference-between-compiled-and-interpreted-language/ [accessed 14th April 2022].
103. Sidana, U. (2021). *Python vs C: Know what are the differences.* Columbia: edureka.co. Available at: https://www.edureka.co/blog/python-vs-c/ [accessed 14th April 2022].
104. Siderova, S. (2018). *The Top 10 Benefits of Kanban.* Unknown: getnave.com. Available from: https://getnave.com/blog/kanban-benefits/ [accessed 23rd May 2022].
105. Simonyan, K. and Zisserman, A., (2014). *Very deep convolutional networks for large-scale image recognition*. arXiv preprint arXiv:1409.1556. Web link: https://arxiv.org/abs/1409resi.1556 [accessed 1st March 2022].
106. Sjursen, S. (2020). *The Pros and Cons of Using Jupyter Notebooks as Your Editor for Data Science Work.* United States: betterprogramming.pub. Available from: https://betterprogramming.pub/pros-and-cons-for-jupyter-notebooks-as-your-editor-for-data-science-work-tip-pycharm-is-probably-40e88f7827cb [accessed 16th May 2022].
107. Smith, S.L., Kindermans, P.J., Ying, C. and Le, Q.V., (2017). *Don't decay the learning rate, increase the batch size*. arXiv preprint arXiv:1711.00489. Available from: https://arxiv.org/abs/1711.00489 [accessed 23rd May 2022].
108. Soni, P. (2022). *Data augmentation: Techniques, Benefits, and Applications.* Available from: https://www.analyticssteps.com/blogs/data-augmentation-techniques-benefits-and-applications [accessed 21st April 2022].
109. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., (2014). *Dropout: a simple way to prevent neural networks from overfitting*. The journal of machine learning research,

15(1), pp.1929-1958. Available from: https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf?utm_content=buffer79b43&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer, [accessed 23rd May 2022].

110. Stapel, F.T.A., (2020). *A Heuristic Approach to Indoor Rock-Climbing Route Generation* (Bachelor's thesis, University of Twente). Available from: http://essay.utwente.nl/80579/1/stapel_BA_eemcs.pdf [accessed 03 October 2021].

111. Surmenok, P. (2017). *Estimating an Optimal Learning Rate for a Deep Neural Network.* United States: towardsdatascience.com. Available from: https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0 [accessed 25th April 2022].

112. Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A.A., (2017). *Inception-v4, inception-resnet and the impact of residual connections on learning*. In Thirty-first AAAI conference on artificial intelligence. Available from: https://arxiv.org/pdf/1602.07261.pdf [accessed 2016].

113. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., (2015). *Going deeper with convolutions*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9). Available from: https://arxiv.org/pdf/1409.4842.pdf [accessed 22nd May 2022].

114. Tai, C.H., Wu, A. and Hinojosa, R., (2020). *Graph Neural Networks in Classifying Rock-Climbing Difficulties*. Technical report. Web link: http://cs230.stanford.edu/projects_winter_2020/reports/32175834.pdf [accessed 03 October 2021].

115. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C. and Liu, C., (2018). *A survey on deep transfer learning*. In International conference on artificial neural networks (pp. 270-279). Springer, Cham. Available from: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5288526 [accessed 15th May 2022].

116. Tan, M. and Le, Q., (2019). *Efficientnet: Rethinking model scaling for convolutional neural networks*. In International conference on machine learning (pp. 6105-6114). PMLR. Available from: https://arxiv.org/pdf/1905.11946.pdf [accessed 22nd May 2022].

117. Tang, P., Wang, X., Shi, B., Bai, X., Liu, W. and Tu, Z., (2018). *Deep FisherNet for image classification*. IEEE transactions on neural networks and learning systems, 30(7), pp.2244-2250. Available from:

https://ieeexplore.ieee.org/abstract/document/8522050 [accessed 12th May 2022].

118. Taqi, A.M., Awad, A., Al-Azzo, F. and Milanova, M., (2018). *The impact of multi-optimizers and data augmentation on TensorFlow convolutional neural network performance*. In 2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR) (pp. 140-145). IEEE. Available from: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8396988 [accessed 16th May 2022].
119. Torrey, L., Shavlik, J., Walker, T. and Maclin, R., (2006). *Advice-based transfer in reinforcement learning*. University of Wisconsin machine learning group working paper 06, 2. Available from: https://www.researchgate.net/profile/Richard-Maclin/publication/250358764_Advice-based_Transfer_in_Reinforcement_Learning/links/53ece1c40cf2981ada10fa6d/Advice-based-Transfer-in-Reinforcement-Learning.pdf [accessed 15th May 2022].
120. Lundh, F. (2011). *Python Imaging Library (Fork).* [Software]. Available from: https://pillow.readthedocs.io/en/stable/index.html [accessed 14th April 2022].
121. Uzair, M. and Jamil, N., (2020). *Effects of hidden layers on the efficiency of neural networks*. In 2020 IEEE 23rd international multi-topic conference (INMIC) (pp. 1-6). IEEE. Available from: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9318195&tag=1 [accessed 29th April 2022].
122. Vijayasarathy, L. and Turk, D., (2012). *Drivers of agile software development use: Dialectic interplay between benefits and hindrances.* Information and Software Technology, 54(2), pp.137-148. Available from: https://www.sciencedirect.com/science/article/pii/S0950584911001807 [accessed 7th May 2022].
123. Walker, N. (2020). *Is a MoonBoard for You?* Canada: gripped.com. Available from: https://gripped.com/indoor-climbing/is-a-moonboard-for-you/ [accessed 14th April 2022].
124. Waskom, M.L., (2021). *Seaborn: statistical data visualization*. Journal of Open-Source Software, 6(60), p.3021. Available from: https://joss.theoj.org/papers/10.21105/joss.03021 [accessed 14th April 2022].
125. Xia, K., Huang, J. and Wang, H., (2020). *LSTM-CNN architecture for human activity recognition*. IEEE Access, 8, pp.56855-56866.

Available from: https://www.researchgate.net/publication/340073805_LSTM-CNN_Architecture_for_Human_Activity_Recognition [accessed 26th April 2022].

126. Xu, Y., Noy, A., Lin, M., Qian, Q., Li, H. and Jin, R., (2020). *WeMix: How to Better Utilize Data Augmentation*. arXiv preprint arXiv:2010.01267. Available from: https://arxiv.org/pdf/2010.01267.pdf [accessed 21st April 2022].
127. Yalçın, G. O. *4 Reasons Why You Should Use Google Colab for Your Next Project.* United States: towardsdatascience.com. Available from: https://towardsdatascience.com/4-reasons-why-you-should-use-google-colab-for-your-next-project-b0c4aaad39ed [accessed 16th May 2022].
128. Yao, L., Mao, C. and Luo, Y., (2019). *Graph convolutional networks for text classification*. In Proceedings of the AAAI conference on artificial intelligence (Vol. 33, No. 01, pp. 7370-7377). Available at: https://ojs.aaai.org/index.php/AAAI/article/view/4725 [accessed 22nd February 2022].
129. You, K., Long, M., Wang, J. and Jordan, M.I., (2019). *How does learning rate decay help modern neural networks?*. arXiv preprint arXiv:1908.01878. Available from: https://arxiv.org/abs/1908.01878 [accessed 23rd May 2022].
130. Zeiler, M.D. and Fergus, R., (2014). *Visualizing and understanding convolutional networks*. In European conference on computer vision (pp. 818-833). Springer, Cham. Available from: https://arxiv.org/abs/1311.2901 [accessed 15th May 2022].