Learn To Climb By Seeing: Climbing Grade Classification with Computer Vision

Kathryn Garcia
Stanford University
kgarcia7@stanford.edu

Julián Rodríguez Cárdenas
Stanford University
julianrc@stanford.edu

Abstract

The MoonBoard is a standardized climbing board popular among the climbing community, with climbing problems being classified into discrete categories. Currently, no perfect classifier exists to determine the grade of a problem. Here, we propose a novel approach toward climbing classifying MoonBoard climbs by providing an image of a climb as input to a vision model. For this, we created a dataset where each climb has a corresponding image where the holds are available. Then, we trained custom-made CNNs and pre-defined models to classify these climbs into the correct difficulty. We achieved performance of 40% on our test dataset, which is comparable to prior non-vision approaches. Since the dataset is significantly imbalanced, we tried to balance the learning by weighting the different classes equally. While balancing the learning that way did not improve the overall results, it resulted in noticeable increases to the average class accuracy. Another novel approach presented here is framing this problem as regression instead of classification: this resulted in slightly worse results, but critically, the off-by-one accuracy (where we also consider a problem as correctly classified when it is put in an adjacent difficulty) consistently improved. Lastly, we show saliency maps indicating what features the models use for classification, as well as a qualitative discussion of the misclassified problems.

1. Introduction

The MoonBoard is a standardized, popular climbing wall used for training and skill development across the global climbing community. In the MoonBoard app, users can select a preset climb based on its difficulty level and associated image. In this project, we propose a deep learning approach using Convolutional Neural Networks to better grade the difficulty of a MoonBoard climb, depending on its relative hold positions and grip types based on the information that is visually presented on the app. The grade given to a climb by an individual user is often very subjective, and the relative difficulty of a climb can be determined by a climber's gender, flexibility, skill-level, height, wingspan, and more. The graded difficulty of MoonBoard climbs are currently determined by community consensus. The grades of the MoonBoard problems range from 6B to 8B+ in the Fontainebleau scale rating system, and we will classify each problem with a specific grade in that range.

In our project, rather than using a $\{0,1\}^{18x11}$ matrix to encode each MoonBoard problem, we directly feed in the image of the MoonBoard problem as an input to our model, so our model can more directly learn the features of the holds on a climb. We will continue to experiment with different CNN architectures for this task.

Overall, we found that using the visual information of the MoonBoard as input to a model performs better than simply using a $\{0,1\}^{18x11}$ matrix with a basic CNN, however, it performs worse than models that also include climbing-specific information as inputs. We also found that simple CNN architectures perform similarly to larger and deeper networks, and posing this problem as a regression task performs similarity to classification.

2. Related Work

In previous literature, researchers have used a variety of ML techniques to determine the graded difficulty of a MoonBoard problem as a classification problem. In one of the early approaches in 2017, Dobles et al. encoded each MoonBoard problem as a $\{0,1\}^{18x11}$ matrix, where each 1 corresponds to the position of a hold on the given climb. They used Convolutional Neural Networks to achieve a 34% accuracy [1]. This paper provides good baseline results, but it is relatively simplistic in its approach. In Duh and Chang's work, after encoding each problem as a $\{0,1\}^{18x11}$ matrix, they use a sequence model to predict the sequence of moves for a climb (i.e. left hand, left hand, right hand). They use the sequence model's output in combination with the hold positions as input to their grade predictor model, which is an LSTM. They achieve 47.5% accuracy. This paper includes domain specific information and achieves good results, but they do not use any of the direct visual information of the MoonBoard layout and holds as information to their models. Kempen also

Table 1. Summary of Literature Review

	Ref [1]	Ref [2]	Ref [3]	Ref [4]
Algorithm	CNN	Sequence + LSTM	GCN	Climbing Symbols + VOMM
Performance	34% Accuracy	47.5% Accuracy	0.73 AUC	64% accuracy for hard vs easy differentiation

includes climbing-specific information as input to models and more specifically, used semi-natural Domain-Specific Language, to encode climbing moves in symbol sequences, which can then be fed into variable-order Markov models (VOMMs). They used one model specifically for classifying easy climbs, and another specifically for hard climbs, and they combined both models to predict if a test climb is easy or hard. They achieved 64% accuracy. However, this paper is limited by its limited classification to easy and hard instead of each climbing grade.

As a different approach by using a different architecture and model, Tai et al. applied Graph Convolutional Neural Networks (GCN) to this problem to achieve an AUC of 0.73. Tai et al. preprocessed the MoonBoard problems into multi-hot vectors, where each route is represented as a 140-dimensional vector where each dimension describes the presence or absence of one of the 140 holds.

Crucially, none of these approaches take a vision-centric approach to solve these problems, with all prior work giving as input to the models some preprocessed information that is not inherently visual. For the first time, we put forward an approach where the input is an image of a climb, for the model to classify its grade.

See Table 1 for a summary of the performance of this prior work.

3. Dataset

Since the MoonBoard website removed the database of problems online, we were unable to scrape the data directly from MoonBoard. We utilized the base dataset available from the MoonBoardRNN project [5] which previously scraped the data available from the MoonBoard website, forming a text-based dataset listing details of different climbs. This included 30642 total climbs of 14 different grades. Each climb includes information, in the form of text, of the start, middle, and end holds, grade, and other metadata irrelevant to this problem.

Climb ID	Grade	Is_benchmark
320671	7A	False
258179	6C	False
306695	6B+	True
334204	7B+	False

Table 2. Example Dataset

While this dataset describes which holds are part of a problem, there is no visual information available to the model. Hence, we created a new dataset from this original dataset. For each climb in the dataset, we generated an image of the climb, by taking an image of the base board, and then circling, with different colors, the start, middle, and end holds. Here is an image of the board, and then an image of the board where we have highlighted the holds.

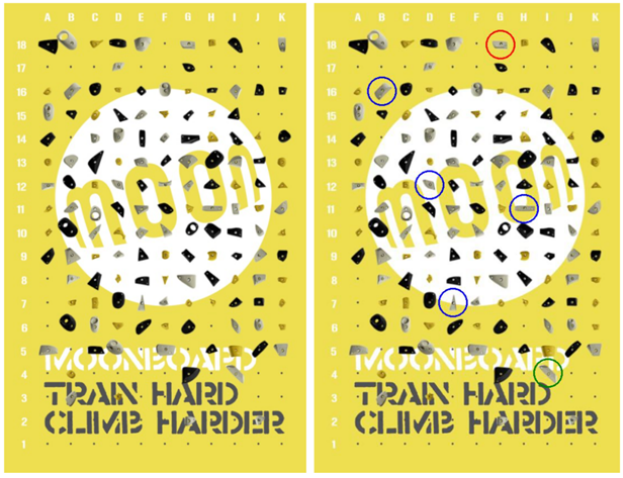

Figure 1. Example of a generated image for a climb

To make the training more streamlined, we also generated a more concise dataset, with only the information needed from each climb: the ID, which is associated with an image of the climb, the grade, and a is benchmark feature (the benchmark climbs have had their quality proved by many people, hence they are ideal candidates for the validation/test datasets). Below are 4 example rows of the new dataset we formed.

Lastly, it is important to note that the 14 classes (14 grades) in the dataset are highly unbalanced, with each higher grade having many less climbs included in the dataset. Here is an exact count of the number of counts of each grade

Grade	Count
6B+	10153
6C+	4617
7A	4045
6C	3836
7A+	3074
7B+	1622
7B	1551
7C	1061
7C+	389
8A	181
8A+	51
8B	29
8B+	24
6B	9

Table 3. Grade Distribution

Notably, upon inspection, we noticed that many climbs in the easiest class and in the two harder classes were of very low quality. That is, the problems did not make sense as members of those classes. This is not entirely surprising, as these climbs are community-made, and there are many reasons why a climb in the dataset may not be up to standards. Thus, we decided to not use those classes for this project, having a total of 12 classes ultimately.

Lastly, to make training more manageable, we downsample our images from 547x847 to 224x224.

4. Methods

The key ingredient in our approach is the usage of images for the climbs, which make the visual features of the holds available to our models. Then, essentially, the classification problem is as follows: a model receives a standard-size image of a climb, where the details of the climb (i.e. which holds are part of the problem) are baked into the picture. Then, the model must output the grade of the climb.

All prior approaches reduced the problems to matrices of 0,1 indicating which holds were available. However, all that would indicate to the model is the relative position of the holds. Even more important than the position, the difficulty of a grade depends upon the type of holds available. Thus, we propose that making the holds visually available to the model will allow it to learn what an easier or harder hold looks like. Then, the model can combine this with information about the positions of the hold to learn how hard a problem is.

As architectures, we decided to use experiment with different models made from scratch using Pytorch, as well as pre-existing models, either pre-trained or not. Overall, the training methodology here is standard. We used 20% of the data for testing. During training, we use batches of images to backpropagate the error. We use cross entropy loss, except for the cases where we do regression instead of classification (see experiments and results), where we use mean squared error.

5. Experiments and Results

Here we list all of our results and experiments.

5.1. Customized CNN

One of our main approaches consisted of training a convolutional neural network from scratched, constructed and trained entirely in PyTorch. The architecture is described and pictured in Figure 1. We picked this specific architecture after some trial-and-error. Simpler networks were unable to learn to classify with any acceptable accuracy, and we found that an architecture like this was complex enough to get an acceptable accuracy.

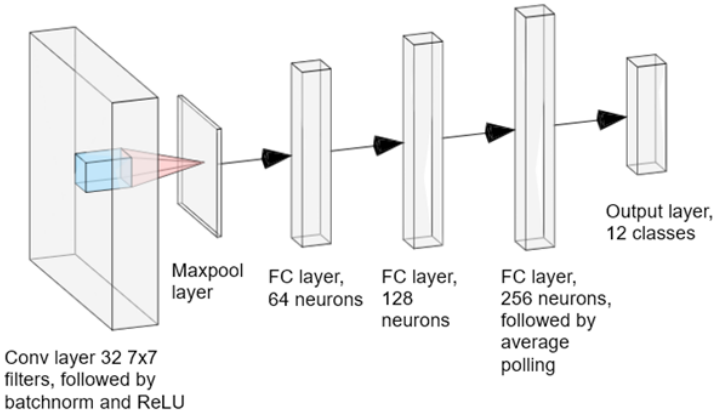

Figure 2. CNN architecture

We tried combinations with different hyperparameters: a batch size from 8, 32, 100, whether to weighting the classes to balance them, and learning rate from 0.0001, 0.00001. The table below reports the validation accuracy after training, as well as the one-off and average class accuracy, using the superior 0.0001 learning rate.

	Weighted True	Weighted False
Batch Size 8	Acc: 41.87% One-off: 73.10% Avg class acc: 22.65%	Acc: 40.62% One-off: 69.68% Avg class acc: 18.39%
Batch Size 32	Acc: 39.25% One-off: 64.73% Avg class acc: 17.33%	Acc: 39.84% One-off: 65.03% Avg class acc: 13.96%
Batch Size 100	Acc: 37.91% One-off: 71.20% Avg class acc: 22.43%	Acc: 41.93% One-off: 69.18% Avg class acc: 19.76%

Table 4. Comparison of Accuracy, One-off Accuracy, and Average Class Accuracy for Different Batch Sizes and Weighting Schemes, Results for classification model, lr = 0.0001, 5 epochs

5.2. Larger Models

In addition to a custom CNN, we used larger, predetermined architectures from PyTorch. These models included resnet50, densenet121, and vit_b_16, to experiment with architectures that had different depths, and to experiment with a vision transformer. To experiment with these models, we used random search to search through of range of hyperparameters. We experimented with these learning rates: [1e-5, 1e-4, 1e-3, 1e-2], these optimizers: [Stochastic Gradient Descent (SDG), 'Adam', 'RMSprop'], batch sizes: [16, 32, 64], and epochs: [4, 6, 8, 10]. Additionally, we experimented with pre-trained models that were trained on IMAGENET1K_V1, and models that had no initial pre-trained weights. For all of these experiments, our primary metrics also included accuracy, one-off-accuracy, and average class accuracy. For these models, we also experimented with and without a weighted loss function to balance our imbalanced dataset. See results in Table 5.

5.3. Weighted Training

The dataset is significantly imbalanced, we often found that the accuracy of the most common class was significantly higher than any others. When training with simpler models, the model would often always just output the most common class. We found that using deeper and more expressive models lead to improvements on the accuracies of other classes, but there was still a clear imbalance (the next Figure shows the accuracy per class of one of our best models).

To ameliorate this problem, we experimented with re-weighting the classes so that each class has the same overall weight during training. We specifically did this using weighted loss functions. As we can see in the figure showing the average class accuracy using this re-weighting, this method increased the accuracy of the less-common examples. Critically, as can be seen in the results table, the overall test accuracy remained stable whether we used this weighting or not, while the average class accuracy saw a notable increase.

	Weighted True	Weighted False
resnet50	Acc: 36.70% One-off: 67.48% Avg class acc: 16.52%	Acc: 37.04% One-off: 68.52% Avg class acc: 17.31%
resnet50 pre-trained	Acc: 37.22% One-off: 69.74% Avg class acc: 19.81%	Acc: 36.70% One-off: 70.43% Avg class acc: 17.82%
densenet121	Acc: 34.96% One-off: 64.87% Avg class acc: 13.93%	Acc: 30.4% One-off: 72.1% Avg class acc: 30.4%
densenet121 pre-trained	Acc: 36% One-off: 68% Avg class acc: 27.02%	Acc: 31.13% One-off: 57.21% Avg class acc: 12.78%
vit_b_16	Acc: 28.42% One-off: 43.13% Avg class acc: 8.3%	Acc: 30.12% One-off: 68.88% Avg class acc: 30.12%
vit_b_16 pre-trained	Acc: 33.22% One-off: 57.22% Avg class acc: 12.45%	Acc: 39.47% One-off: 70.43% Avg class acc: 17.16%

Table 5. Comparison of Accuracy, One-off Accuracy, and Average Class Accuracy for different model architectures. This table shows the best results for each model.

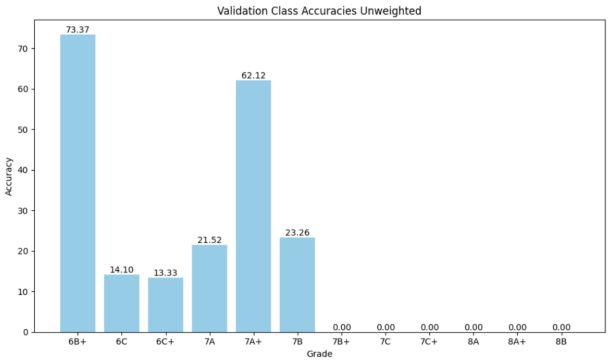

Figure 3. Unweighted Accuracies Across Grades for DenseNet121

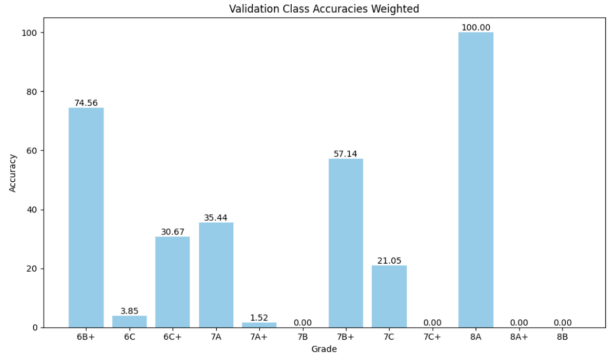

Figure 4. Weighted Accuracies Across Grades for DenseNet121

5.4. Regression vs. Classification

Our default approach framed this as a classification problem using cross-entropy loss. However, since the difficulty of the problems follows a single dimension, we decided to also try framing this as regression problem, having the model output a single number between [1, 12], taking the nearest integer number as the difficulty. The table below shows the results of using the exact same architecture and training as our customized CNN approach, but having a single output unit instead of 12, one for each class. We note that while the overall accuracy is worse, the one-off accuracy is slightly better.

	Weighted True	Weighted False
Batch Size 8	Acc: 34.68% One-off: 73.29% Avg class acc: 34.68%	Acc: 32.18% One-off: 71.66% Avg class acc: 32.18%
Batch Size 32	Acc: 32.08% One-off: 73.39% Avg class acc: 32.08%	Acc: 32.38% One-off: 73.15% Avg class acc: 32.38%
Batch Size 100	Acc: 29.52% One-off: 69.62% Avg class acc: 29.52%	Acc: 27.93% One-off: 66.48% Avg class acc: 27.93%

Table 6. Comparison of Accuracy, One-off Accuracy, and Average Class Accuracy for Different Batch Sizes and Weighting Schemes, Results for regression model, lr = 0.0001, 5 epochs

5.5. Saliency Maps

We used saliency maps to better understand how our models were making their predictions, and to analyze which input features were contributing the most to our model's outputs. These plots show how the most input features changed based on the epoch. We can see that for a given climb, at first, the model locates the holds of interest, and in particular it seems focused on the most difficult part of the climb. Later, it seems like the model has captured most of the holds of interest that are used on the particular climb. As the model continues training however, it starts to use too many features and expands beyond the holds that are directly a part of the climb, leading to poor predictions.

Figure 5. An Example Moonboard Problem

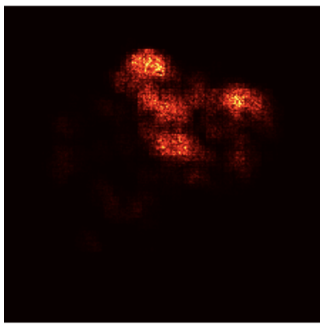

Figure 6. DenseNet121 Not Pretrained: Epoch 2

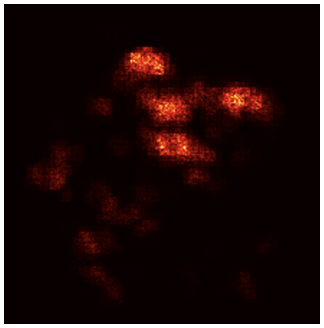

Figure 7. DenseNet121 Not Pretrained: Epoch 4

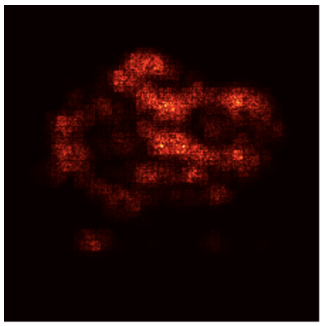

Figure 8. DenseNet121 Not Pretrained: Epoch 6

5.6. Misclassification Analysis

We analyzed some of the misclassified images for certain models to further analyze our predictions. We found one pattern that when there are many holds, our models struggled to predict the correct class, and it seemed like the model may have recognized a pattern that if there are many holds, the route should have a easier grade than it really does. The reverse seems true when there are few holds. It seemed like the models predicted that a climb wold be more difficult if there were fewer holds further away, even though the climb may still be easy.

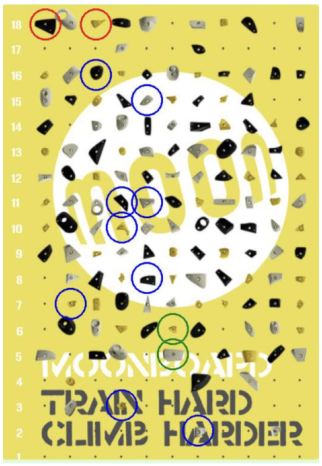

Figure 9. Misclassification example 1: True Label: 6C+, Predicted Label: 6B+, for densenet121 without pre-training.

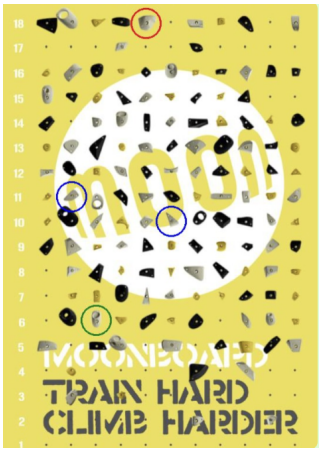

Figure 10. Misclassification example 2: True Label: 6C, Predicted Label: 7B+, for densenet121 without pre-training.

6. Discussion

Model Architecture: Overall, our results have shown that a simplified custom CNN architecture performs just as well, if not better to larger and deeper architectures to the the task of grade classification with rock climbing images on the MoonBoard. This may be because it may be difficult to learn more features of the individual hold difficulties, and how that impacts the complex movement that makes one climb harder than another, even with a more powerful model. For the larger models, it seems like resnet50 and densenet121 model architectures perform similarly across metrics, and the vision transformer performs the worse. This may be because a vision transformer is too powerful for the relatively small amount of training data that we have. Another limitation is that we had limited training time, and there is the chance that with more fine-tuning and a greater training time, the larger architectures may perform better. We also found that pre-training the models on ImageNet did not significantly improve accuracies; this may be because the features of various images on ImageNet are unrelated to the specific features of a climbing hold and position.

We found that it is important to include the one-off accuracy and average class accuracy as metrics to this problem, which many other papers do not include. One-off accuracy is an important metric because the grading of climbs is biased, and climbs of different grades may actually close in difficulty, especially for climbers of different body types and abilities. Also, It is a more difficult task to predict the grades of more difficult climbs, and it is important that that is highlighted in the validation metrics by using average class accuracy. We want our models to be suited to classify routes in the whole range of climbing grades.

Weighted Training: We have found that weighted training using a weighted loss function is important for this particular classification task, since our classes are so imbalanced. In the results above, we can see that weighted training improves average class accuracies, and more difficult grades with few examples receive much higher accuracies. Other papers such as [2], use weighted training, and for our approach, it also is important for the class imbalance.

Related Works: In comparison to other related works, our results perform better to a base CNN [1] that uses a $\{0,1\}^{18x11}$ grid as an input for training. It seems like our models are able to learn more from the visual information of the climbing board. However, our metrics are worse than approaches that incorporate climbing specific information as inputs to their models, such as climbing sequence information (which holds follow each other in the sequence of climbing movement) [2]. Grade classification is a very specific task, and there is a lot of information that a climber needs to appropriately grade a climb, that is not captured in just an image of the climbing board.

Regression: Overall, using regression instead of classification did not improve the results. This was a surprising result, since this problem is well suited to regression; after all, difficulty here is a single dimension. We also note that all the prior work framed this as a classification problem, which we found similarly surprising. Our hypothesis for why classification did just as well as regression, is that fundamentally the model does learn a regression-esque structure. Across all of our models, the one-off accuracy is significantly higher than the regular accuracy (sometimes the one-off accuracy is double the regular one). In other words, when a model makes a mistake about the grade of a problem, it is often the case that it misclassified it as slightly easier or slightly harder, which is a similar kind of mistake we would expect to see in regression. Then, we conclude that doing classification still allows the model to properly learn that difficulty works along a single dimension.

The visual approach: Is the classification of Moon-

Board problems well suited to computer vision? We emphasize that one novel approach in this project is showing images as inputs to the network, instead of matrices of 0, 1. Indeed, no one before seems to have used any actual images for classifying these climbing problems, and part of the issue is that the board always looks the same. Two problems are different not because the board is set up differently, but because different holds are allowed. Thus, to signal to the model what the problem is, we circled the holds. This worked, as our test-accuracy was just as good as those of prior attempts, and often better. However, we wonder if there are better ways of indicating to the model what the allowed holds are. We also wonder if ultimately this problem is well-suited for computer vision. Normally, when two images are classified as different, the images are different; here, besides the circles indicating the holds, all climbs look exactly the same.

Image size: Lastly, one expectation we had of the model was that it would learn what a hard and easy hold looks like, and then use that knowledge to asses the difficulty of a climb. However, subtle visual differences in holds often make a significant difference to the climber, and when the images are down-sampled to 224x224, it is possible that crucial detail is lost. We believe that a worthwhile approach for this problem is use larger resolution images.

7. Conclusions and Future Work

Overall, grade classification of climbing images is a difficult task. It is difficult for neural networks to capture all of the nuances of different rock climbs and how they contribute to its difficulty. In our project, we have shown that the visual information of the MoonBoard image itself can be used for this classification tasks, to achieve similar results to previous literature. However, previous work that includes additional climbing-specific information achieves better results. We have also learned that classification is just as suited as regression for this task, and weighted training is important to account for the imbalanced dataset.

In the future, we would suggest experimenting with higher resolution images for this visual approach, and if using more powerful architectures, potentially increasing the size of the dataset and more time for greater fine-tuning of models. Going forward, it would also be interesting to combine the climbing-specific information (such as climbing sequence) methods with the visual information of the MoonBoard.

8. Contributions and Acknowledgements

For this project, both team members developed the research question, experiments, and analysis together. Kathryn worked specifically to develop and experiment with the larger model architectures, and worked to produce qualitative results, such as saliency maps and misclassification analysis. Julian worked to develop the data pipeline by generating the custom images for our dataset, and worked specifically to develop and experiment with the custom CNN architecture.

We both divided up the rest of the work.

9. References

[1] Alejandro Dobles, Juan Carlos Sarmiento, and Peter Satterthwaite. Machine Learning Methods for Climbing Route Classification. Technical report, 2017.

[2] Y.-S. Duh and R. Chang. Recurrent Neural Network for MoonBoard Climbing Route Classification and Generation. ArXiv:2102.01788 [Cs], 2021. https://arxiv.org/abs/2102.01788

[3] Cheng-Hao Tai, Aaron Wu, and Rafael Hinojosa. Graph Neural Networks in Classifying Rock Climbing Difficulties.

[4] Lindsay Kempen. A fair grade: assessing difficulty of climbing routes through machine learning, 2019.

[5] Chang, R. (2022, May 20). MoonBoardRNN. GitHub. https://github.com/jrchang612/MoonBoardRNN

[6] S. Imambi, K.B. Prakash, and G.R. Kanagachidambaresan. PyTorch. In: K.B. Prakash and G.R. Kanagachidambaresan (eds) Programming with TensorFlow. EAI/Springer Innovations in Communication and Computing. Springer, Cham, 2021. https://doi.org/10.1007/978-3-030-57077-4_10

[7] O. Kramer. Scikit-Learn. In: Machine Learning for Evolution Strategies. Studies in Big Data, vol 20. Springer, Cham, 2016. https://doi.org/10.1007/978-3-319-33383-0_5

[8] A. Rastogi. Visualizing Neural Networks using Saliency Maps in PyTorch. Medium, April 7, 2020. https://medium.datadriveninvestor.com/visualizing-neural-networks-using-saliency-maps-in