DeepClimb: CNN for Automatic Bouldering Route Difficulty Assessment

Gael Colas
Aeronautics and Astronautics department
Stanford University
colasg@stanford.edu

Peter Satterthwaite *
Electrical Engineering
Stanford University
psatt99@stanford.edu

Abstract

Assessing the difficulty level of a climbing route is a hard and subjective problem, even for professional climbers. This paper expands on the work of Dobles et al. [2]. In particular, we propose the first deep learning approach using visual information to classify climbing routes. Two main approaches were considered: the first one uses an end-to-end CNN architecture working directly from route images. The second approach proposes using a CNN encoding stage to build "hold embeddings", a vector representation of holds, from hold images. The data was scrapped from the MoonBoard platform which provides a database of climbing routes set on the same standardized wall. The end-to-end CNN approach outperforms the previous state-of-the-art set by Dobles et al. [2]: our model achieves 40% accuracy on the test set. These results validate the usefulness of working with visual representation. This intuition is supported by an analysis of saliency maps, that indicates that the network learns to recognize the correct regions of interest: the available holds.

1. Introduction

Assessing the difficulty level of climbing routes is a very important task. First, climbers use these grades to evaluate their progress, and find inconsistent grades detrimental to their learning experience. In addition, it is primordial for safety reasons: a climber can avoid climbing routes that he is not prepared for by relying on the grading system. This paper focuses on a subset of climbing routes, namely bouldering routes. Bouldering routes are shorter climbing routes usually ascended without ropes. Historically, several grading systems have been proposed by the climbing community to evaluate the difficulty of bouldering routes. The Fontainebleau system was selected in this paper because it is the most widely used internationally. In this system, bouldering routes are graded on an alphanumeric scale according to the following format "number-letter-sign". The number scale ranges from 1 to 9 in an increasing order of difficulty. Within a number level, the following letter (A, B or C) ranks the routes in increasing order of difficulty. Finally, a + sign can be added afterwards to give some more granularity (a 7C+ should be more difficult than a 7C, but less than a 8A).

Unfortunately, despite a set grading system, assessing the difficulty grade of a specific route is a highly subjective task. This paper aims to build a CNN classifier to automatically assess the level of a bouldering route from its picture. The end goal is to provide a consistent grading tool for the climbing community. This paper works on a well-defined subset of bouldering routes: our dataset is composed of bouldering routes set on the MoonBoard wall. The MoonBoard is a standardized bouldering wall with a fixed set of holds. It can be found in many climbing gyms around the world. The company that sells it released a platform so that MoonBoard's users can share the route they set with climbers around the world.

2. Related Work

There has not been much work done on applying Machine Learning tools to climbing. Phillips et al. [8] published the most cited paper on the topic. They proposed a system, called "Strange Beta", to assist the process of route setting. Their model combines machine learning techniques with a chaos variation generator to design novel climbing routes. The authors claim that setters using their software outperform setters who do not in terms of route quality.

Kosmalla et al. [6] proposed a system to identify climbing routes using sensors worn by the climber. A set of inertia units wrapped at the climber's wrists records measurements during the ascent. Then their model extracts and uses features from the recording to identify the route that has been climbed. However this system does not provide any indication about the difficulty of the route. Kempen [5] proposed "a semi-natural Domain-Specific Language" representation for climbing routes. Each symbol of this lan-

*Peter Satterthwaite is not enrolled in CS231n. He provided the starter code for the web-scraper he wrote for [2].

guage is a climbing move. A route can thus be represented as a sequence of symbols. The grade classification task was framed as a variable-order Markov models. Unfortunately, their best model did not outperform the naive strategy of predicting the most common class. Finally Dobles et al. [2] worked on classifying bouldering MoonBoard routes. They compared the performance of several Machine Learning models on this task: a Naive Bayes classifier, a Softmax classifier and a CNN network. Their dataset was only composed of routes from the 2016 version of the MoonBoard. As the set of holds and their positions is fixed for all these routes, their models use a binary representation of the routes (see Section 3 for more details). They framed this classification as an ordinal classification [3]: the idea being that the labels are ordered by increasing difficulty.

This paper builds mostly on the work from Dobles et al. [2]. To the author's knowledge, this paper presents the first attempt at incorporating visual information to assist the classification task.

3. Data

3.1. Data source

Data was sourced from the MoonBoard website: `moonboard.com`. The MoonBoard is a standardized bouldering wall which can be found in many climbing gyms around the world. It consists of a set of up to $n = 198$ holds placed on a 18x11 grid. On the website, a bouldering route is described by a String listing its characteristics. The useful characteristics for this project are: the list of available holds, the grade (the difficulty level of the route assigned by the setter) and the user grade (the difficulty level of the route assigned by the community) if there is one. The list of available holds is the subset of holds a climber can use to climb the wall, from given start holds to end holds.

3.2. Data description

There are currently two main versions of this wall: a 2016 version ("MoonBoard 2016") and a 2017 one ("MoonBoard Masters 2017"), each with its dedicated set of holds. **At the time of the scrapping, the 2016 and the 2017 version accounted for 26,489 and 18,306 different routes respectively.**

Each route is associated with a difficulty grade determined by its setter. This is the label for our models. Difficulty levels range from 6A+ to 8B+, we have $C = 15$ ordinal classes. The dataset is very imbalanced: more than 30% of the routes have the easiest grades (less than 6B+) while less than 1% of them are graded 8A+ or more. Figure 1 shows the grade distribution for the two versions of the MoonBoard.

In addition, 38% of the routes were also assigned a grade by the community. This grade is the average of the grades assigned by the climbers who tried the routes. It can be considered as the ground truth, but cannot be used for labeling as less than half of the routes have community assigned grades. However, this provides a good estimate of the noisiness of the dataset's labels: the dataset's labels are not very noisy, the user and the community assigned grades coincide 97% of the time.

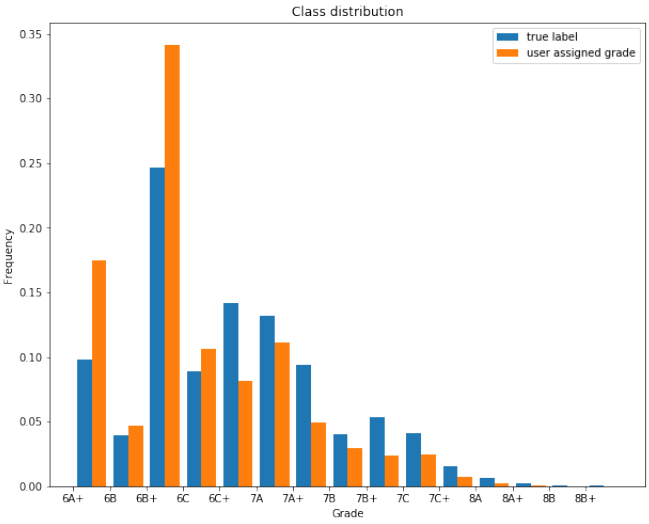

Figure 1: Class distribution for the whole dataset. In blue: the true grade distribution. In red: the user grade distribution for the subset of the routes that have a grade assigned by the community.

For each version, the dataset of routes was randomly split according to a 80% / 10% / 10% train / validation / test split. For the 2016 version, the routes are divided into 21,733 training routes, 2,725 validation routes and 2,725 test routes. For the 2017 version, the routes are divided into 16,025 training routes, 2,009 validation routes and 2,009 test routes. A careful attention was put in preserving the overall grade distribution in every split.

3.3. Pre-processing pipeline

Since for each version the position of the holds is fixed, a hold can be equivalently described by its position on the grid. So a bouldering route can be represented as a binary matrix of size 18x11 where each component is a binary feature indicating whether or not the hold can be used to climb the wall. The route can also be visualized as an RGB image of the MoonBoard wall where the available holds are circled, see Figure 2 for an example.

A pre-processing pipeline was built to convert the scraped String description of a route into the binary representation. Then the second stage of the pre-processing was to convert this binary representation into the RGB image

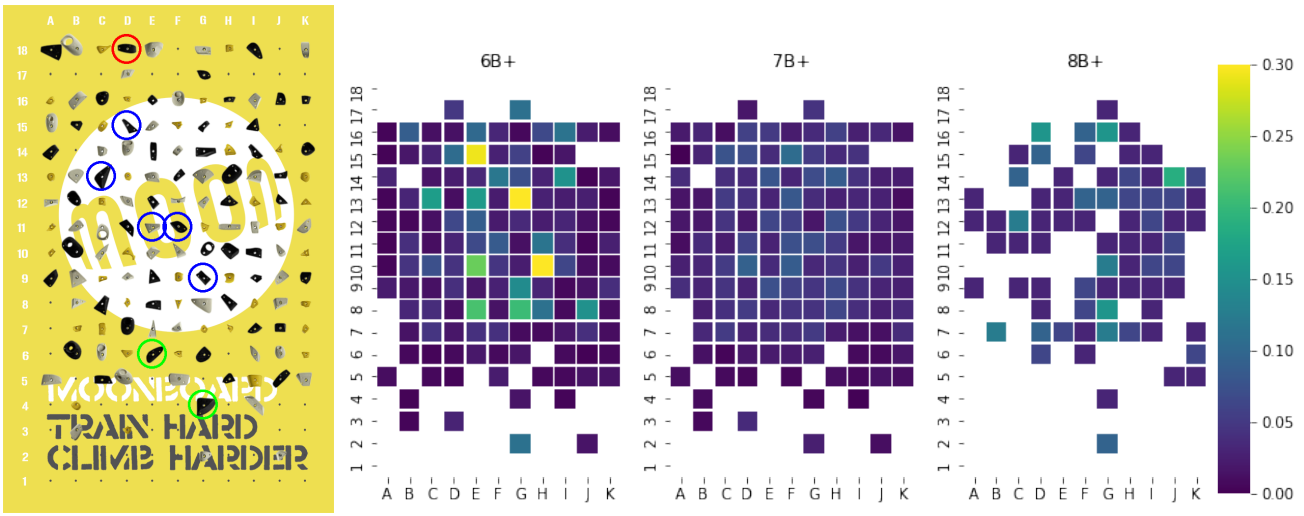

Figure 2: Left: image representation of a route on the 2016 MoonBoard wall. Right: heatmap indicating the frequency each hold is used as an intermediate hold in easy (6B+), medium (7B+) and hard (8B+) routes. We can see that both easy and hard routes have a characteristic set of holds that are used more often. For intermediate routes, we can see that all the holds are used equally. This suggests it might be trickier to distinguish between intermediate grade labels using the binary matrix.

representation. For each version of the MoonBoard, a blank image of the wall was used as a template: an image of the whole wall with all the holds. Then for each example, an "artificial" RGB image representation was generated from its binary representation and its version's template image. The available holds were circled on the template image after being extracted from the binary matrix. The following color code was used to account for different types of holds: green for start holds, blue for intermediate holds and red for end holds.

The same dataset split (see Section 3.2) was used for both representations to allow for model comparisons.

3.4. Inputs

The way to represent a bouldering route depends on the method used. As proposed in Dobles et al. [2], for the Bernoulli Naive Bayes model, each route was represented as a binary matrix of size 18x11. A component of this matrix is set to 1 if the corresponding hold can be used for this route, 0 otherwise. For this representation, we needed the set of holds to be fixed. Thus for these models, only the dataset composed of routes set on the 2016 version of the MoonBoard was used.

For the Convolutional Neural Networks (CNN), to incorporate the visual information of the available holds, we use the RGB image representation. An input is an RGB image of the MoonBoard where the available holds are circle, see Figure 2. Each image is of size 1,000 x 650 pixels (height x width). Finally, to reduce the computational complexity of handling big images while retaining the shape information of the holds, the images were resized to 364 x 256 pixels (height x width). The aspect ratio was preserved. The images were normalized by subtracting the mean pixel (computed over the training data). Finally a random left-right flip was used to perform data augmentation. For the CNN models, adding the visual information of the available holds allows to work with both datasets (the training sets of both versions are merged).

4. Methods

4.1. Bernoulli Naive Bayes

As a baseline, a Naive Bayes classifier with a Bernoulli event model was first trained. For a given example the binary features of this model are the holds' indicator functions from the example's binary representation.

As Dobles et al. [2] explained in their work, the Naive Bayes assumption is violated in our problem: the binary features are not independent given the label. Indeed, some holds on the wall work as pairs: for example, when one is available for the right hand, the other is often available for the left hand.

The other issue is that because the dataset is unbalanced, the Naive Bayes classifier will be biased towards the most common class. To deal with this unbalance, at train time the examples are weighted according to their class label. The following weighting scheme was used:

$$w_j = p_j^{-k}, 1 \leq j \leq C$$

Where: C is the number of classes, $p_j = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}(y^{(i)} = c)$

is the true class distribution, and $k \geq 0$ is a hyperparameter.

The weight of an example is inversely proportional to the proportion of examples of its class at the power k. For $k = 0$, we have the Vanilla Bernoulli Naive Bayes. Increasing k increases the importance of examples from less common classes. The hyperparameter k was tuned on the validation set. $k = 6$ was found to give the best validation accuracy.

4.2. Convolutional Neural Network

Dobles et al. [2] only used the binary representation to train their classifiers. However meaningful information is lost in this representation: harder routes tend to be comprised of smaller holds which are harder to grab. So the shapes of the available holds matter when doing a grade classification.

A natural next step was to use a CNN architecture to incorporate the visual information from the image representation of routes. Two main approaches have been considered:

- **End-to-end learning**

The first approach is to train a CNN to predict the grade label directly from the image representation.

The route images are not square and because it is important to preserve the aspect ratio, they cannot be resized as squares. Otherwise, meaningful information will be lost: the distance between successive holds and the shape of the holds in particular. Due to this constraint and the absence of prior work, using an existing CNN architecture was not an option.

Several architectures were tried, Table 1 summarizes the selected custom architecture tuned on the validation set. The network is composed of 8 successive convolution blocks. A convolution block is composed of a convolutional layer followed with a Batch-Normalization layer (momentum $\rho = 0.1$), a ReLU activation layer and a Max-Pooling layer. All the convolutional layers are padded to preserve the input height and width dimensions. The Max-Pooling layer shrinks the height and width of the activation by 2 (with a 2x2 filter and a stride of 2). The last convolution block replaces the Max-Pooling layer with an Average-Pooling over a filter of shape 3 x 2, to account for the non-square aspect ratio. Finally a 1x1 convolutional layer shrinks the number of channels to match the number of output classes, the output is resized to a column vector of logits.

Table 1: End-to-end CNN architecture. CONV F-N-POOL denotes a convolutional block: the succession of a convolutional layer, a Batch-Norm layer, a ReLU layer and a pooling layer. The convolutional layer has N filters, each of size FxFxD, where D is the depth of the activation volume from the previous layer, padding is same, and stride is 1. The pooling layer uses the POOL method.

Layer	Filter Dimensions	Activation Dimensions
INPUT	None	384 x 256 x 3
CONV3-8-MAX	8 x 3 x 3 x 3	192 x 128 x 8
CONV3-8-MAX	8x 3 x 3 x 8	96 x 64 x 8
CONV3-16-MAX	16 x 3 x 3 x 8	48 x 32 x 16
CONV3-16-MAX	16 x 3 x 3 x 16	24 x 16 x 16
CONV3-32-MAX	32 x 3 x 3 x 16	12 x 8 x 32
CONV3-32-MAX	32 x 3 x 3 x 32	6 x 4 x 32
CONV3-64-MAX	64 x 3 x 3 x 32	3 x 2 x 64
CONV3-64-AVG	64 x 3 x 3 x 64	1 x 1 x 64
CONV1-15	15 x 1 x 1 x 64	15

- **"Hold embedding"**

In the first approach, the network has to learn by itself to pay attention to the circled holds and use their shape information to guide the prediction. This might prove to be too difficult, which justifies this different approach. A first stage small CNN builds a vector representation ("hold embedding") from the cropped images of each available hold. Then this hold embedding is concatenated to the indicator function of the binary representation. This is the new 3D input of the CNN architecture.

The advantage of this model is that the useless visual information from the unavailable holds is filtered out. The network does not have to identify by itself which are the available holds.

The CNN models were trained with a Cross-Entropy loss computed from the output logits (before applying the softmax function). The equation of the Cross-Entropy loss is:

$$CE = -\frac{1}{m}\sum_{i=1}^{m}\sum_{1\leq j\leq C} y_j^{(i)} \log(\text{softmax}(o^{(i)})_j)$$

Where: m is the number of examples, C is the number of classes, $y^{(i)}$ is the label of the i-th example (one-hot encoded), and $\text{softmax}(o^{(i)})$ is the softmax of the i-th example's logits as computed by the model.

5. Experiments

Two distinct types of experiments were run. For the models using the binary representation, a fixed set of holds is needed. Indeed, we want the indicator function of an example to refer to the same hold from one example to another. Thus only the 2016 version dataset was used to train and evaluate these models. The CNN models using the image representation can deal with various holds' configurations as they are able to look at the holds. Thus both the

2016 and the 2017 version dataset were used to train and evaluate these models (their train, validation and test sets were merged).

5.1. Evaluation methods

The models are compared according to the following metrics on the validation set:

- Accuracy (Acc): both overall and within classes.

$$Acc = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}(y^{(i)} = \hat{y}^{(i)})$$

  Where: m is the number of examples, $y^{(i)}$ is the label of the i-th example, and $\hat{y}^{(i)}$ is the predicted label of the i-th example from the model.

- Mean Absolute Error (MAE): for this problem, there is a clear ordering of the classes. Thus it makes sense to measure the distance between the predicted and the true label. At equal accuracy, the model with the smallest MAE is the best one.

$$MAE = \frac{1}{m}\sum_{i=1}^{m} |y^{(i)} - \hat{y}^{(i)}|$$

- KL-divergence (KL): for this problem, we have a very imbalanced dataset (see Figure 1 for more details). The Kullback-Leibler (KL) divergence measures how well the model captures the underlying true class distribution, or if on the contrary the model is biased towards certain classes.

$$KL(p, \hat{p}) = \sum_{1 \leq j \leq C} p_j \log(\frac{p_j}{\hat{p}_j})$$

  Where: C is the number of classes, $p_j = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}(y^{(i)} = j)$ is the true class distribution, and $\hat{p}_j = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}(\hat{y}^{(i)} = j)$ is the predicted class distribution from the model. $\log$ is the natural logarithm.

5.2. Results

Table 2 summarizes the validation performance of the models. Our first model is Naive Bayes. It uses the binary representation, so it was evaluated on the validation set of the 2016 version. All the models are compared with the most naive approach ("Naive"): always predict the most common class (6B+ for the 2016 version). The table also indicates the performance of the best model ("Softmax") trained by Dobles et al. [2]: a Softmax classifier taking as input the binary representation of routes. The performance of their model was given on an unreleased test set, however the source as well as the distribution of this test set are the same as our validation set.

Table 2: Model performance on the validation set. For the models using the binary representation, the validation set of the 2016 version. For the CNN model, the merged validation sets of the 2016 and 2017 versions. The best results are indicated in bold.

Model	Accuracy	MAE	KL-divergence
Naive	32.0 %	2.44	Inf
Softmax	36.5 %	1.37	**0.078**
Naive Bayes ($k = 6$)	37.1 %	1.6	0.08
End-to-end CNN	**39.9** %	**1.17**	0.12

The end-to-end CNN model uses the image representation, so it was evaluated on the merged validation sets of the 2016 and 2017 versions.

The end-to-end CNN is our best model: it achieves the best validation accuracy and mean absolute error. Its final performance was evaluated on the test set. It is nearly identical to the validation performance, which indicates a good generalization. **The end-to-end CNN achieves a 40.0% accuracy, a 1.21 mean absolute error and a 0.12 KL-divergence on the test set**. Our best model outperforms by around 4% the accuracy of the best model from Dobles et al. [2].

Figure 3 describes the test performance of the end-to-end CNN per class. The plot on the right compares the true class distribution with the distribution of predicted labels from the model. We can see that the model captures the trend of the true class distribution. However as expected, because of the dataset imbalance, some classes are not predicted at all by the model. In particular, grades higher than 8A+ (which represent less than 1% of the whole training set) are never predicted.

The model makes a mistake more than half of the time. However the mean absolute error indicates that the misclassifications often happen close to the true label: on average around 1 class away. This is especially relevant in our setup as there is a clear ordering of the classes. As a consequence, our model can be used to provide a first guess of what the grade of a route should be. This is confirmed by the confusion matrix. The matrix elements far from the diagonal are almost all zeroes. Most of the mistakes come from misclassifications between neighboring classes. For example, the 6B and 6C are always misclassified as 6B+. This comes from the fact that in our dataset, 6B and 6C appear far less than the most common class 6B+.

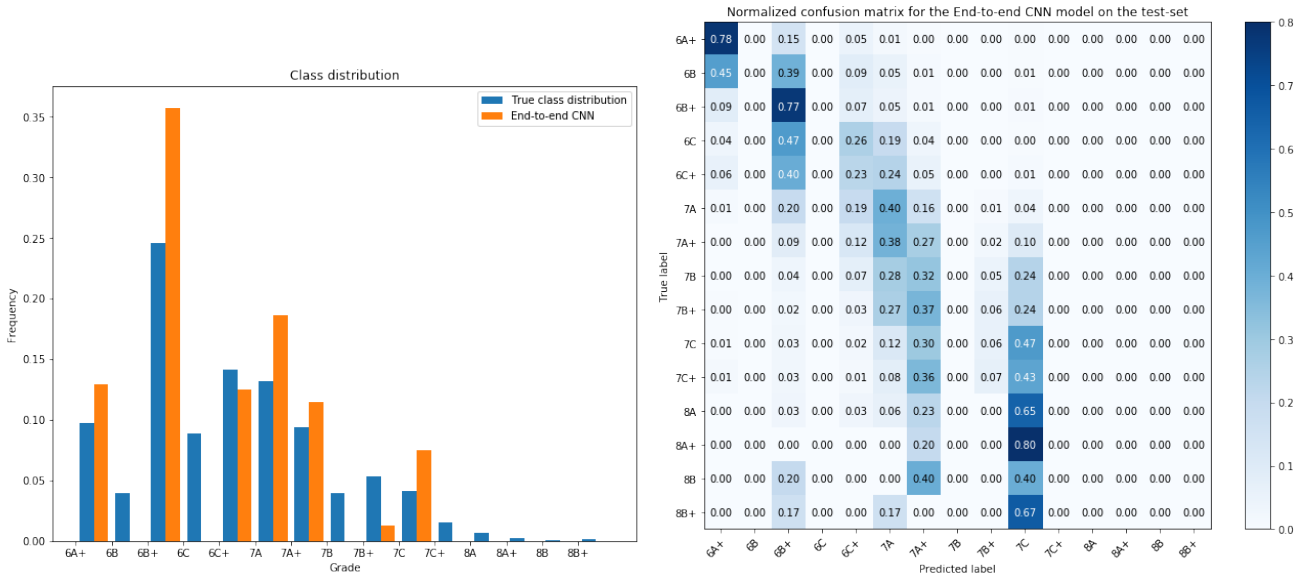

Figure 3: Test set performances of the end-to-end CNN. Left: comparison between the true class distribution (in blue) and the predicted class distribution (in orange). Right: normalized confusion matrix.

5.3. Analysis

The CNN model was trained with an Adam optimizer. The initial learning rate was tuned on the validation set to provide a stable learning: $\alpha = 0.01$ was found to provide the fastest initial loss decrease while being robust to divergence. The training then plateaued rapidly. A cosine learning rate schedule over 50 epochs was found to help decrease the loss further. Finally, L2-regularization was used to remedy to the initial overfit observed. At the end of the training, the best model losses were: 1.59 for the training set and 1.61 for the validation set. This small gap indicates a small overfit. Table 3 describes the training hyper-parameters in more details.

Table 3: Training hyper-parameters

Hyper-parameter	Value
Optimizer	Adam
Initial learning rate (α)	0.01
Number of epochs	50
Mini-batch size	64
Learning rate schedule	cosine
L2-regularization (λ)	0.01

To make its prediction, the end-to-end CNN must first learn to identify the available holds (the ones circled on the image), then use their shape information to infer the grade of the route. To evaluate if the model was succeeding at the first step, saliency maps were computed for a random subset of the validation examples. Figure 4 shows two examples (one for each MoonBoard version) with their corresponding saliency maps. The pixels that most influence the loss are the one with the highest loss gradient. From the saliency maps, we can see that these pixels' locations match the inside of circled regions. This means that the network is looking at the right spots.

6. Conclusion

The end-to-end CNN model outperforms the previous state-of-the-art set by Dobles et al. [2]: our model achieves 40% accuracy on the test set. Moreover, even if the model still makes a mistake more than half of the time, the misclassifications happen close to the true objective (with a mean absolute error around 1). Our model offers a good first guess of the route grades. This legitimates the usefulness of working with an image representation of routes, instead of only using their binary representation. This intuition is corroborated by the analysis of saliency maps that indicates that the network learned to correctly identify the regions of interest: the available holds. Our model also has the advantage of being robust to different configurations of the MoonBoard. Indeed, when using the binary representation, a fixed configuration is needed to ensure that each binary feature has the same meaning for every example. However, when working with the image representation, this constraint disappears as the network can now distinguish between holds from their images.

Due to the time constraint, the "hold embedding" approach was not studied. It would be interesting to see fur-

Figure 4: Validation set examples with their saliency maps computed with the end-to-end CNN model. Left: an example from the 2016 version of the MoonBoard. Right: an example from the 2017 version of the MoonBoard. In the saliency maps, white pixels indicate high absolute loss gradient with respect to this pixel of the input image.

ther work pursuing this promising idea. First, the end-to-end approach studied in this paper might have proved to be too complicated due to the limited size of the dataset. Filtering out the information of unused holds with the "hold embedding" approach could help alleviate the problem. The "hold embedding" approach would also allow to work with higher resolution images of holds. This could be a good way to evaluate the usefulness of the holds' shape information for the prediction.

In the future, it would be interesting to generalize the CNN approach to more diverse types of bouldering walls. Ideally, the end goal would be to design an API that allows to take a photo of any bouldering wall, select the holds you can use and then run the CNN classification network to predict the grade of the route. Such application would be very useful to the climbing community to enforce a consistent grading system. Unfortunately, the absence of a more diverse dataset of bouldering routes prevents such a system to be trained.

Source Code

This project's code can be found in this GitHub repository: `https://github.com/ColasGael/DeepClimb`.

This project is implemented in PyTorch [7] version 1.0.

Contributions

Peter Satterthwaite provided the starter code for the web-scraper he wrote in [2]. Gael Colas implemented and wrote everything else. Starter codes were used for some part of the implementation. They are described in Acknowledgement.

Acknowledgement

We would first like to thank the CS231n teaching staff for their help throughout the quarter, in particular for providing the code to set up Google Cloud [4]. We would also like to thank the CS230 and the CS224n teaching staff for respectively providing an example code for an image DataLoader [9] and a training script in PyTorch [1]. These files were used as a starter code for this project.

References

[1] Chris Chute (CS224n teaching staff). *Example of a training script in PyTorch*, 2019 (accessed in May 2019).

[2] A. Dobles. Machine learning methods for climbing route classification. 2017.

[3] P. A. Gutierrez, M. Perez-Ortiz, J. Sanchez-Monedero, F. Fernandez-Navarro, and C. Hervas-Martinez. Ordinal regression methods: survey and experimental study. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):127–146, 2016.

[4] Jim Fan (CS231n teaching staff). *How to setup Google Cloud (Jupyter and CUDA for PyTorch)?*, 2019 (accessed in May 2019).

[5] L. Kempen. A fair grade: assessing difficulty of climbing routes through machine learning. 2018.

[6] F. Kosmalla, F. Daiber, and A. Krüger. 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*, CHI '15, pages 2033–2042, New York, NY, USA, 2015. ACM.

[7] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[8] C. Phillips, L. Becker, and E. Bradley. Strange beta: An assistance system for indoor rock climbing route set-

ting using chaotic variations and machine learning. *CoRR*, abs/1110.0532, 2011.

[9] Surag Nair (CS230 teaching staff). *Example of DataLoader for images in PyTorch*, 2018 (accessed in May 2019).