MACHINE LEARNING CLIMBING GRADES

MSCA 32019 REAL-TIME INTELLIGENT SYSTEMS PROJECT (WINTER 2020)

BY LAURA TOCIU

TABLE OF CONTENTS

- BACKGROUND AND SIGNIFICANCE
- MOONBOARD GRADING PROBLEM
- SOFTWARE ARCHITECTURE
- CUSTOM CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE
- ACCURACY
- NEXT STEPS

SETTING CLIMBING ROUTES IS TIME-CONSUMING

- In climbing gyms, it is VERY hard to set problems economically and efficiently:
  - Limited inventory of holds of different shapes and sizes
  - Different setters may have preferences for certain holds, reducing the availability for other setters
  - Different setters have different climbing styles, meaning they will assess difficulty differently
  - Setting process involves screwing and unscrewing holds to move them to achieve appropriate distances/angles
- Outdoors, it is EVEN HARDER to set routes:
  - There is vast amount of rock
  - Setting process involves rappelling from top and drilling metal into wall, often **before** knowing for sure the climb will be good or even climb-able

Data science could help make setting in gyms and recognizing promising rock outdoors **less time-consuming and more economical**

BACKGROUND AND SIGNIFICANCE

PREVIOUS DATA SCIENCE APPLIED TO CLIMBING IS SCARCE

- Chaos 2012 Mar; 22(1):013130 – Mathematics of chaos and Variational-Order Markov Model to aid gym route setters in creating novel routes
- https://fmt.ewi.utwente.nl/media/30-TScIT_paper_46.pdf
- Iot Technologies for Healthcare - Book Chapter

} Predicting grades from sequence of movements (recorded by wearables or climber-reported)

- https://cmougan.github.io/files/TFM.pdf
- https://arxiv.org/pdf/2009.13271.pdf

} Using Autoencoders to encode and generate Moonboard problems

- CS230 Stanford final project report – Use Graph Neural Network to grade Moonboard problems
- CS229 Stanford final project report
- Luke321321 Github

} Use Convolutional Neural Network to grade Moonboard problems

- Andrew Houghton Github -- Use LSTM, MLP, Random Forest to grade Moonboard problems + cool problem generator

BACKGROUND
AND SIGNIFICANCE

PREVIOUS DATA SCIENCE APPLIED TO CLIMBING IS SCARCE

- Chaos 2012 Mar; 22(1):013130 – Mathematics of chaos and Variational-Order Markov Model to aid gym route setters in creating novel routes
- https://fmt.ewi.utwente.nl/media/30-TScIT_paper_46.pdf
- Iot Technologies for Healthcare - Book Chapter

} Predicting grades from sequence of movements (recorded by wearables or climber-reported)

- https://cmougan.github.io/files/TFM.pdf
- https://arxiv.org/pdf/2009.13271.pdf

} Using Autoencoders to encode and generate Moonboard problems

- CS230 Stanford final project report – Use Graph Neural Network to grade Moonboard problems
- CS229 Stanford final project report
- Luke321321 Github

} Use Convolutional Neural Network to grade Moonboard problems

- Andrew Houghton Github -- Use LSTM, MLP, Random Forest to grade Moonboard problems + cool problem generator

BACKGROUND
AND SIGNIFICANCE

WHAT IS THE MOONBOARD?

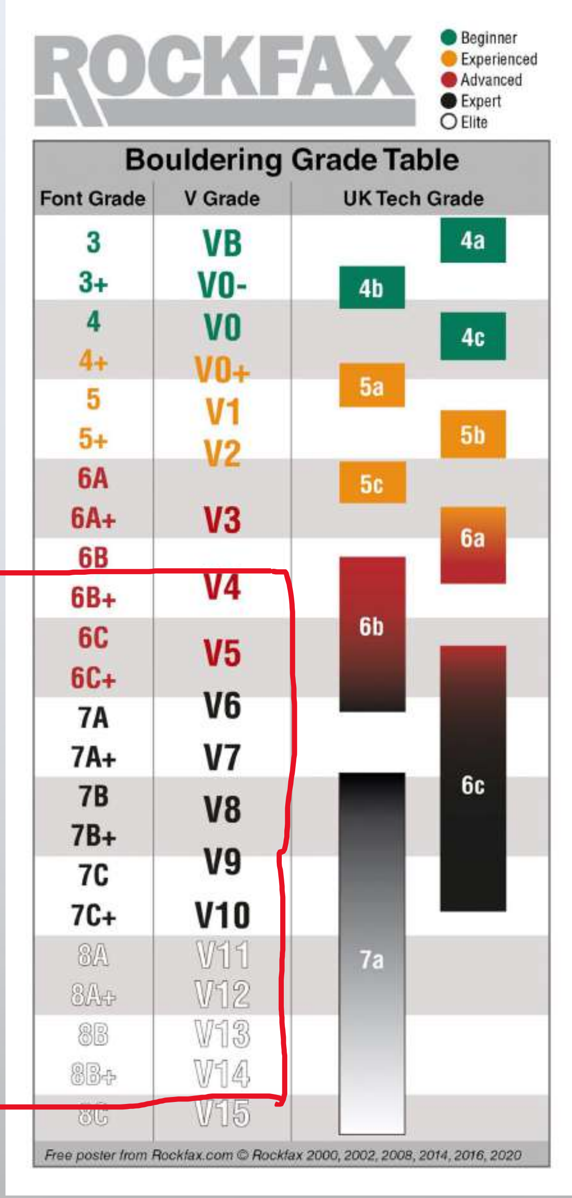
“The MoonBoard is a standardised interactive training wall that connects a global community of climbers through shared problems and competitive performance rankings.”

Powered by

- Moonboard app, where users upload problems (sequence of holds)
- LED lights, that light up the holds whenever a user selects a problem

Users climb routes set by others and give them a grade

CLIMBING GRADES



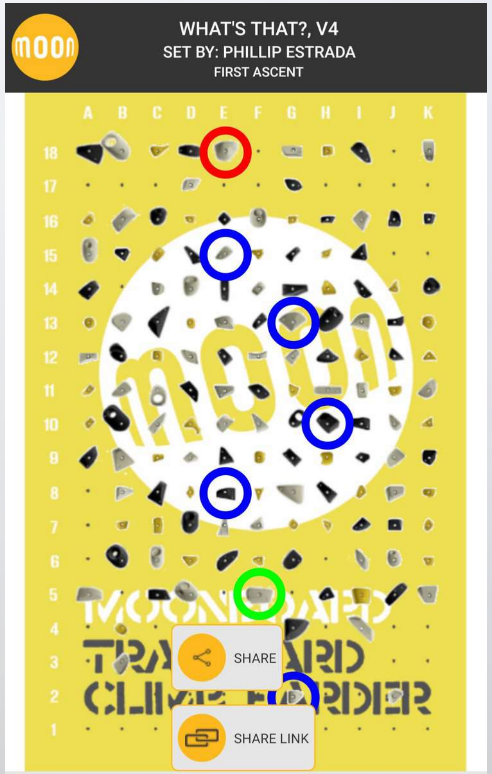
Grades typically on Moonboard

13 categories on Font scale

11 categories on V scale

MOONBOARD GRADING PROBLEM

WHAT IS THE PROBLEM I WANT TO TACKLE, AND WHY?



Map Moonboard problem pictures to grades

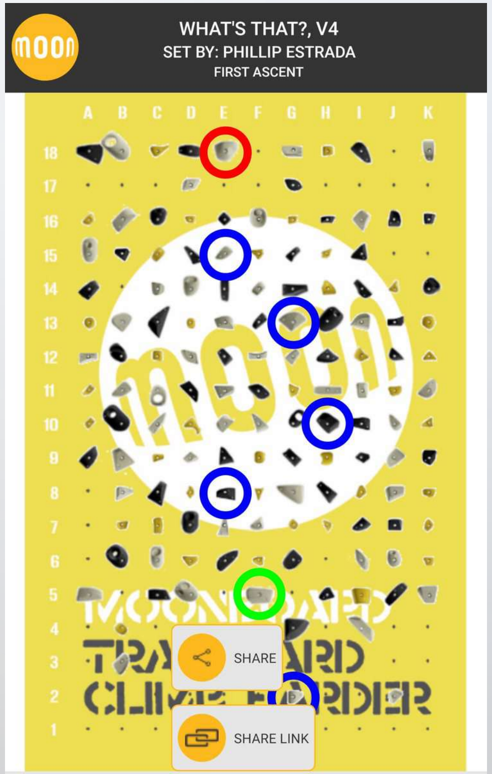
using Convolutional Neural Networks (CNN)

V4

Why the Moonboard grading problem?

- This is the **most approachable** data science problem that can be tackled in climbing
    - Moonboard app has tens of thousands of problems that can be webscraped
    - The problems have been graded by many users so they should be consistent
- The problem **still has not been solved** with great accuracy
- Solving it accurately can pave the way to grading more complex climbing routes and to generating even better routes automatically using AI

WHAT IS THE PROBLEM I WANT TO TACKLE, AND WHY?



Map Moonboard problem pictures to grades

using Convolutional Neural Networks (CNN)

V4

Why using CNN?

- Solving this problem using only pictures is not easy
- CNN will be used as a **starting point** for exploring what can be achieved
- CNN is the **first option tool** when it comes to image recognition and classification

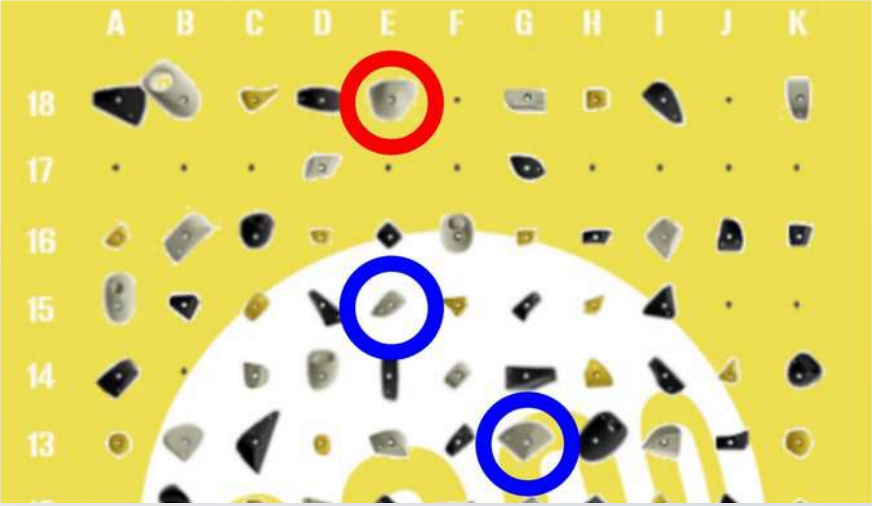
SOFTWARE ARCHITECTURE LAYOUT

The project was implemented using Python 3

Elements:

- A database of 25877 Moonboard problems (link), pre-processed, divided in train/test sets
- Processing script to prepare the data as input for machine learning
- Machine learning model
- Script that evaluates performance of the model

DATABASE CONTAINS LOCATION OF HOLDS AND GRADE



Holds circled are E18, E15 and G13

Relevant database tables and columns

Table: problemMoves

	Problem	Position
	Filter	Filter
1	341208	K18
2	341208	K16
3	341208	F15
4	341208	B11
5	341208	A5
6	341208	H15
7	341206	F5
8	341206	H10
9	341206	E10
10	341206	G13
11	341206	E15
12	341206	G17
13	341206	E18
14	341206	F8

Table: problems

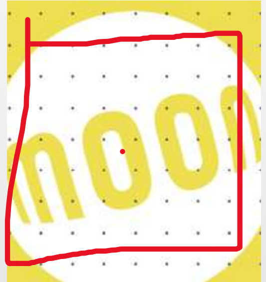
	Id	Name	Grade
	34120	Filter	Filter
1	341208	OFF THE MATTR...	8A+
2	341206	COCO2	6B+

CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE

Issue 1

Applying CNN blindly does not work

Small filters can't capture relevant surroundings



Large filters → many trainable parameters in next layers → network gets stuck due to small dataset (25000)

Solution:

Use large filters but only perform convolution when a hold exists at the center (reduces size of next layers)

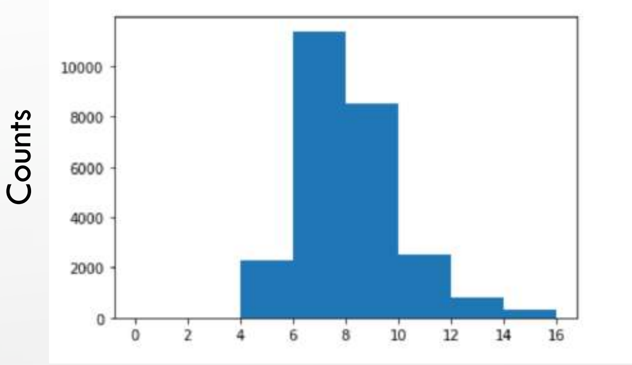
Inspired by [CS229 Stanford final project report]

ML MODEL

Issue 2

Different problems have different number of holds

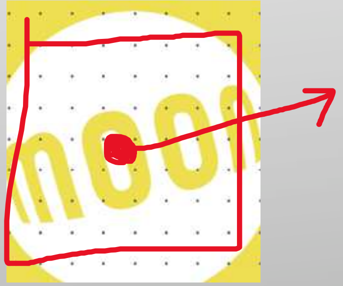
Upsample each problem so all problems have 12 holds total



Issue 3

Just environment around hold is ALSO not enough.

Solution: Feed some rudimentary information on each hold to the network, which is possible using the convolution only at the center model

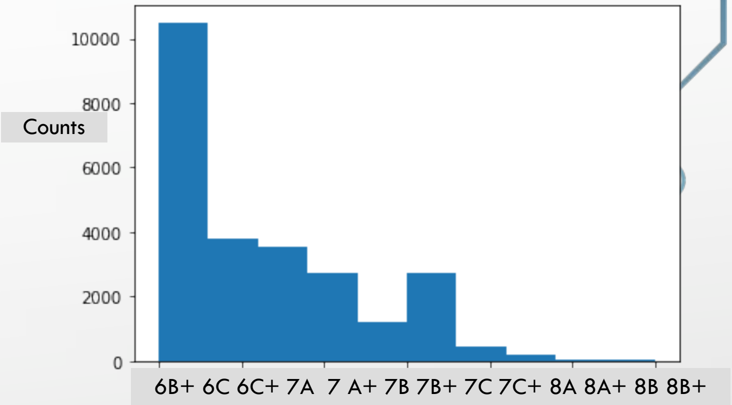


Easy
Medium
Hard?

Neighbors will be considered neutral (medium)

Issue 4

Imbalanced dataset



Solution:

Combine pro-level climbs (7C and above) in one category

Issue 5

The labels are ordered, meaning all grades under V4 are easier than V4 etc.

Regular **softmax activation** that is used for classification is **not appropriate**.

Solution:

Output is [0,0,0… 0]

[1,0,0…,0]

…

[1,1,1,…,1]

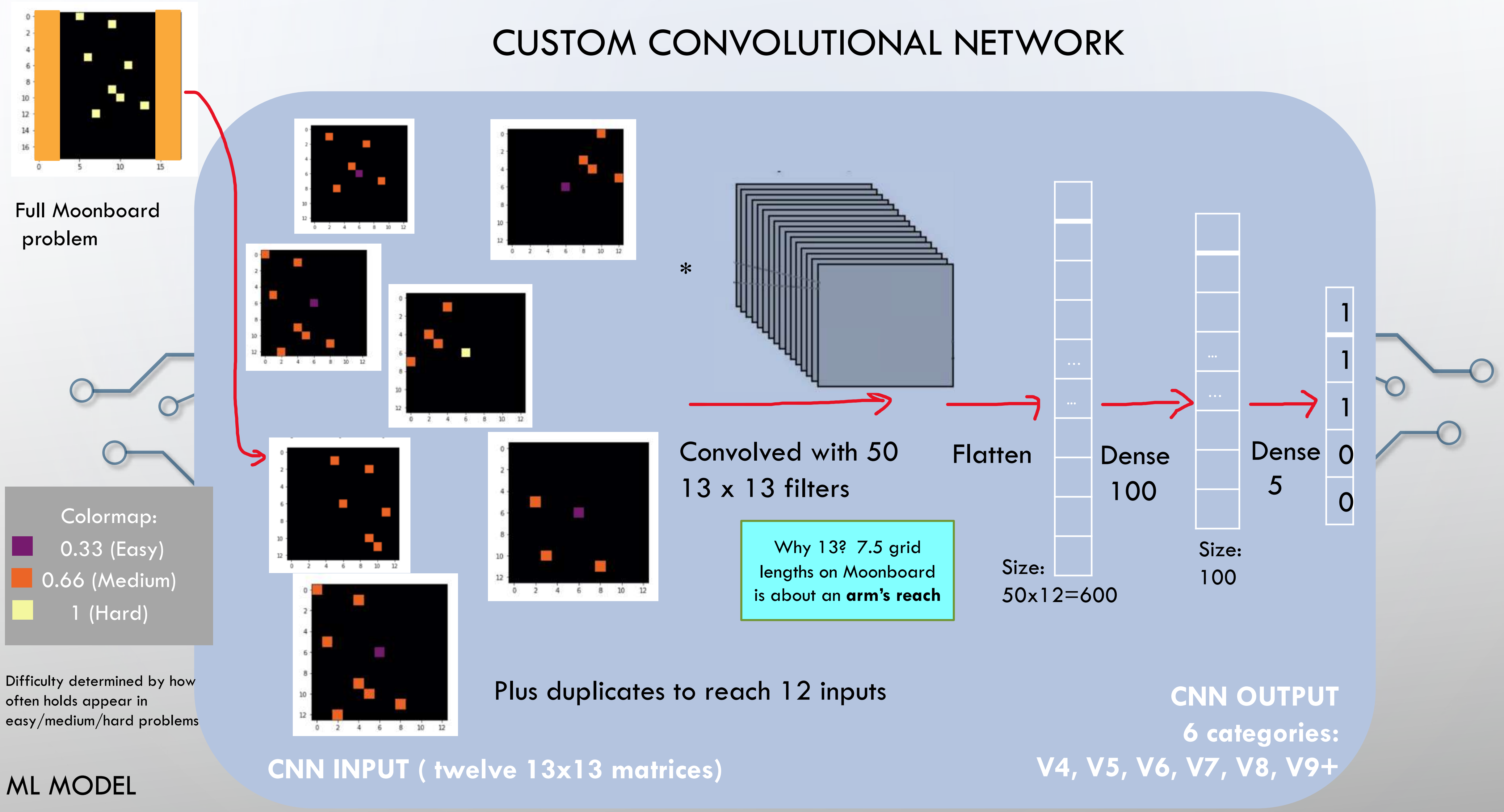
N-1 output neurons for N categories

Activation is sigmoid function, and probabilities are converted to binary for each neuron

Source:

https://ieeexplore.ieee.org/abstract/document/4633963

CUSTOM CONVOLUTIONAL NETWORK



ML MODEL

SNAPSHOTS OF CODE

```
db = 'moon_problems.db'
conn = sqlite3.connect(db)
c = conn.cursor()
c.execute("SELECT pm.Problem, pm.Position, pm.isStart, pm.isEnd, p.Grade FROM problemMoves_2016 pm INNER JOIN problems
problem_grade = {}
problem_moves = {}
problem_lengths = {}
problem_lengths_list = []
problem_moves_grade = {}
id_current = 0
id_previous = 0
holds = []
for entry in c.fetchall():
    id_previous = id_current
    id_current = entry[0]
    if id_previous != id_current:
        problem_moves[id_previous] = holds
        holds = []
        problem_grade[id_current] = entry[4]
    holds.append(entry[1])
problem_moves[id_current] = holds
```

SNAPSHOTS OF CODE

```
class Conv2D_centered(Layer):
    def __init__(self, kernel_shape):
        super(Conv2D_centered, self).__init__()
        self.kernel_shape = kernel_shape

    def build(self, input_shape):
        # Create a trainable weight variable for this layer.
        self.kernel = self.add_weight(name='kernel',
                                      shape=self.kernel_shape,
                                      initializer='uniform',
                                      trainable=True)
        self.shape = input_shape
        super(Conv2D_centered, self).build(input_shape)  # Be sure to call this at the end

    def call(self, matrix):
        filtered = []
        for m in range(self.kernel_shape[0]):
            filtered.append(tf.reduce_sum(tf.reduce_sum(matrix*self.kernel[m], axis=-1), axis=-1))
        filtered_concatenated =  tf.concat(filtered, axis=1)
        return filtered_concatenated

    def compute_output_shape(self, input_shape):
        return (input_shape[0], input_shape[1]*self.kernel_shape[0])
```

```
def define_model():
    model = Sequential()
    model.add(Conv2D_centered((50,13,13)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_regularizer=regularizers.l1(0.02)))
    model.add(Dense(10, activation='sigmoid'))
    # compile model
    model.compile(optimizer='adam', loss= 'bce', metrics=['mse'])
    return model

X_train, X_validation_test, Y_train, Y_validation_test = train_test_split(problems, grades_ml, test_size=0.3)
X_validation, X_test, Y_validation, Y_test = train_test_split(X_validation_test, Y_validation_test, test_size=0.66)

model = define_model()
history = model.fit(problems_train, grades_train, epochs=200, batch_size=64, verbose=2)
```

ML MODEL

SNAPSHOTS OF CODE

```
print(model.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_centered_3 (Conv2D_ce (None, 550)               8450
_________________________________________________________________
flatten_3 (Flatten)          (None, 550)               0
_________________________________________________________________
dense_6 (Dense)              (None, 100)               55100
_________________________________________________________________
dense_7 (Dense)              (None, 5 )                1010
=================================================================
Total params: 64,560
Trainable params: 64,560
Non-trainable params: 0
_________________________________________________________________
None
```

ML MODEL

Quick recap of accuracy metrics for multi-class classification models

If you have 4 categories (a, b, c, d) and two sets: real_values and predicted_values

Accuracy = num of times they match / total samples → not good metric when imbalanced dataset

Actual Classes

Predicted Classes	a	b	c	d
a	50	3	0	0
b	26	8	0	1
c	20	2	4	0
d	12	0	0	1

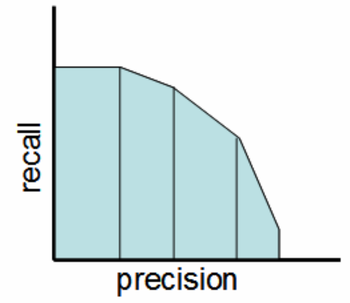
Confusion matrix

$$\mathrm{Precision}(class = a) = \frac{TP(class = a)}{TP(class = a) + FP(class = a)} = \frac{50}{53} = 0.943$$

$$\mathrm{Recall}(class = a) = \frac{TP(class = a)}{TP(class = a) + FN(class = a)} = \frac{50}{108} = 0.463$$

$$\text{F-1 Score} = \frac{2 \times \mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

0 <= F-1 score <= 1



AUC (area under curve) <= 1

Existing literature using a variety of machine learning tools achieve, typically:

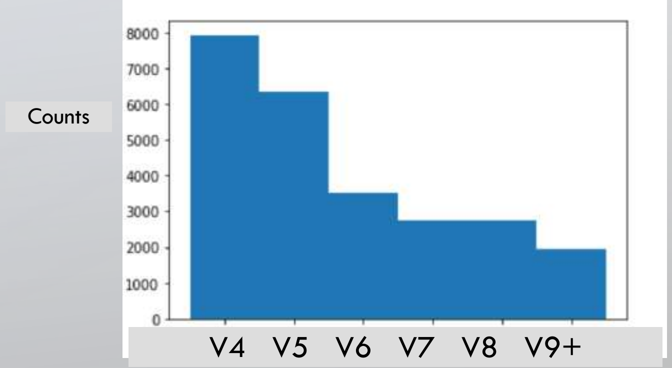
- Accuracy 30-40%
- Average F-1 score 0.2-0.3
- AUC of about 0.7

on classifying Moonboard problems into 10-13 categories

My best solution, presented here, achieves:

- Accuracy 42%
- Average F-1 score 0.34
- AUC of about 0.71

on classifying Moonboard problems into 6 categories



Random guessing would give 1/6

Guessing V4 all the time would give 31%

ACCURACY

HOW DOES THE SOLUTION COMPARE TO OTHER CNN IN LITERATURE?

Only one well-documented work has analyzed CNN for Moonboard problems before ([CS229 Stanford final project report](#))

The main difference is that they do not feed information on central hold (EASY, MEDIUM, HARD)

To compare directly, my code would have to classify the Moonboard problems into 11 categories, not 6

When the code is run for 11 categories, the accuracy is decreased to about 40%

The reported accuracy in the Stanford paper is 34%

The addition of feeding rudimentary difficulty information on the central hold improves the best accuracy found for CNN's in previous literature!

4.2 Convolutional Neural Network Classifier

In light of recent progress in image classification using convolutional neural networks (CNN) [4], a CNN was implemented to classify route difficulty. The implemented CNN is shown in Figure 2. The features for this network were the un-flattened hold matrices $\in \{0,1\}^{18\times 11}$. One convolutional layer with stride one was implemented having four filters of size 11x7 (2.2mx1.4m on the moonboard), and one filter of size one which fed hold information directly to the next layer. Convolution was only performed when filters were centered on a hold, and set to zero otherwise, so that the filters looked at the context of each hold. The subsequent hidden layer of size $5(18 \times 11)$ had a sigmoid activation function and flattened the information from the convolutional layer. The second fully connected hidden layer of size 50 also used sigmoid activation functions. The output layer used an ordinal regression as the activation function. With k categories, the hypothesis function $h(x^{(i)}) \in [0,1]^k$ of the ordinal regression can be described as follows [8]:

$$h(x^{(i)})_j = p(y^{(i)} \leq j) - p(y^{(i)} \leq j-1) \qquad (2)$$

This takes advantage of the ordering of the categories being fit, knowing, for example, that routes of grade 1,2,3,4 are all easier than a route of difficulty 5. The individual probabilities are calculated as follows:

$$p(y^{(i)} \leq j) = \begin{cases} 0, & j = 0 \\ s(b_j - w^T x^{(i)}), & 0 < j < k \\ 1, & j = k \end{cases} \qquad (3)$$

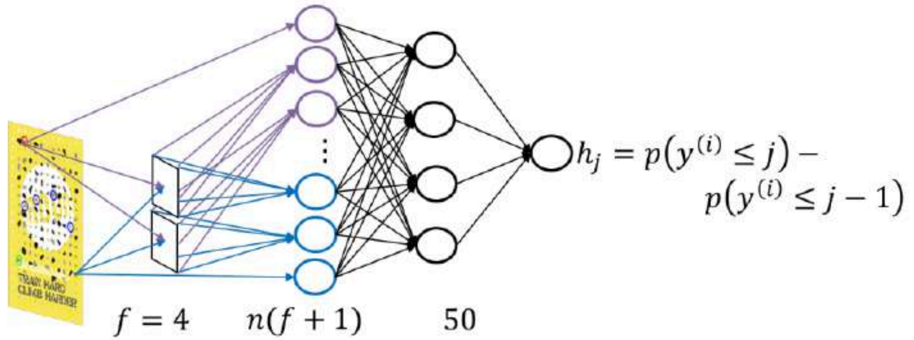
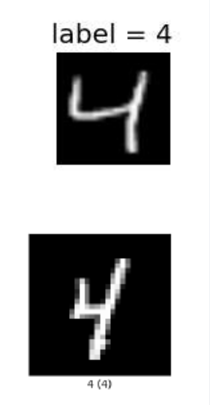


Figure 2: Diagram of implemented convolutional neural network. Example problem is shown on the left.

ACCURACY

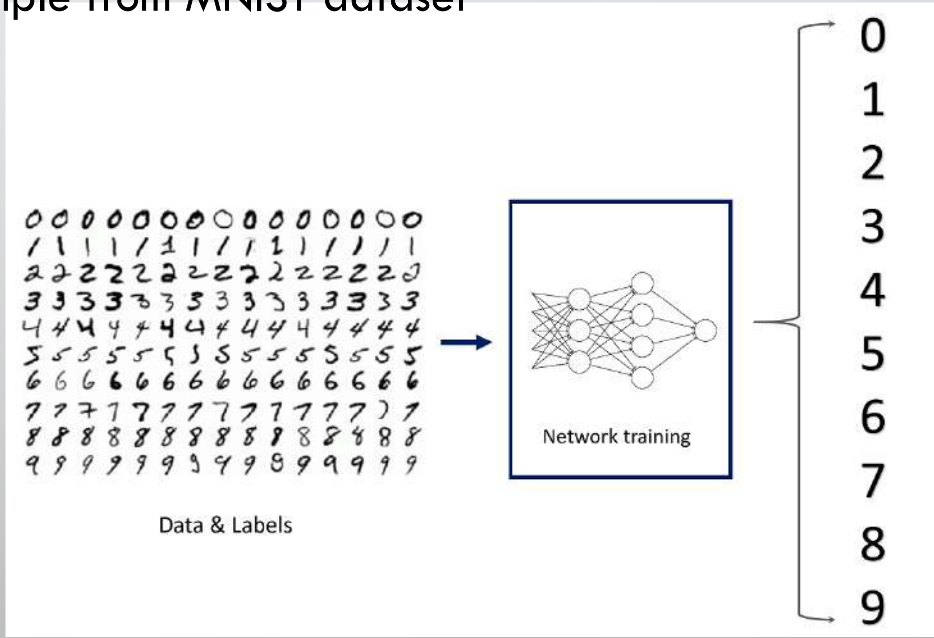
WHY CONVOLUTIONAL NEURAL NETWORKS (CNN) PERFORM POORLY ON MOONBOARD DATA

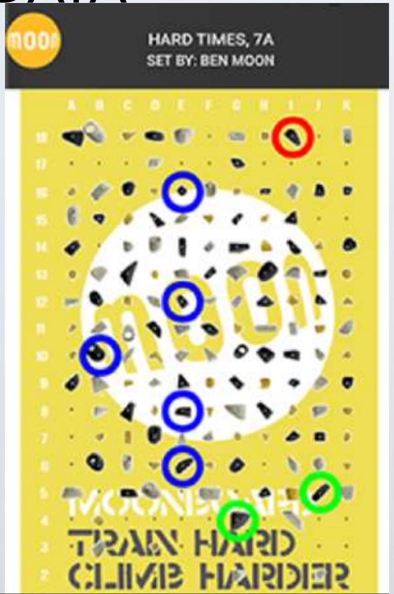


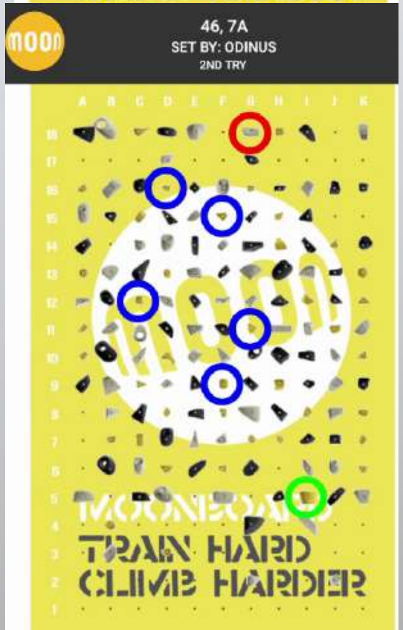
Similarities in

- edge lengths
- angles
- curves

Example from MNIST dataset







Do not have many similarities that are encoded purely in 2D images

- The 3D shape of the holds determines how easily they can be grabbed
- Even if an entire 3D image is used, predicting grade of unseen problems is a hard problem, since new combinations of holds may appear than in training set and the difficulty is not a simple sum of holds or moves

ML MODEL

FUTURE STEPS

- Improve machine learning architecture to achieve better accuracy, using novel architectures, loss functions, constraints etc
- Attempt to webscrape new problems added to the website and grade them in real time
    - Design parallel codes that can access the website from multiple IP's to avoid time-out issues
    - Scrape through all active users and all problems and identify which problems were added that day – will require long run-time, stable connection etc
    - Connect to MySQL server and add the new problems, along with the predicted grade, to the moonboard-grades.db database
    - Periodically check users gradings to compare with predicted grades and improve model
- Use grading model to aid in generative models to create new problems