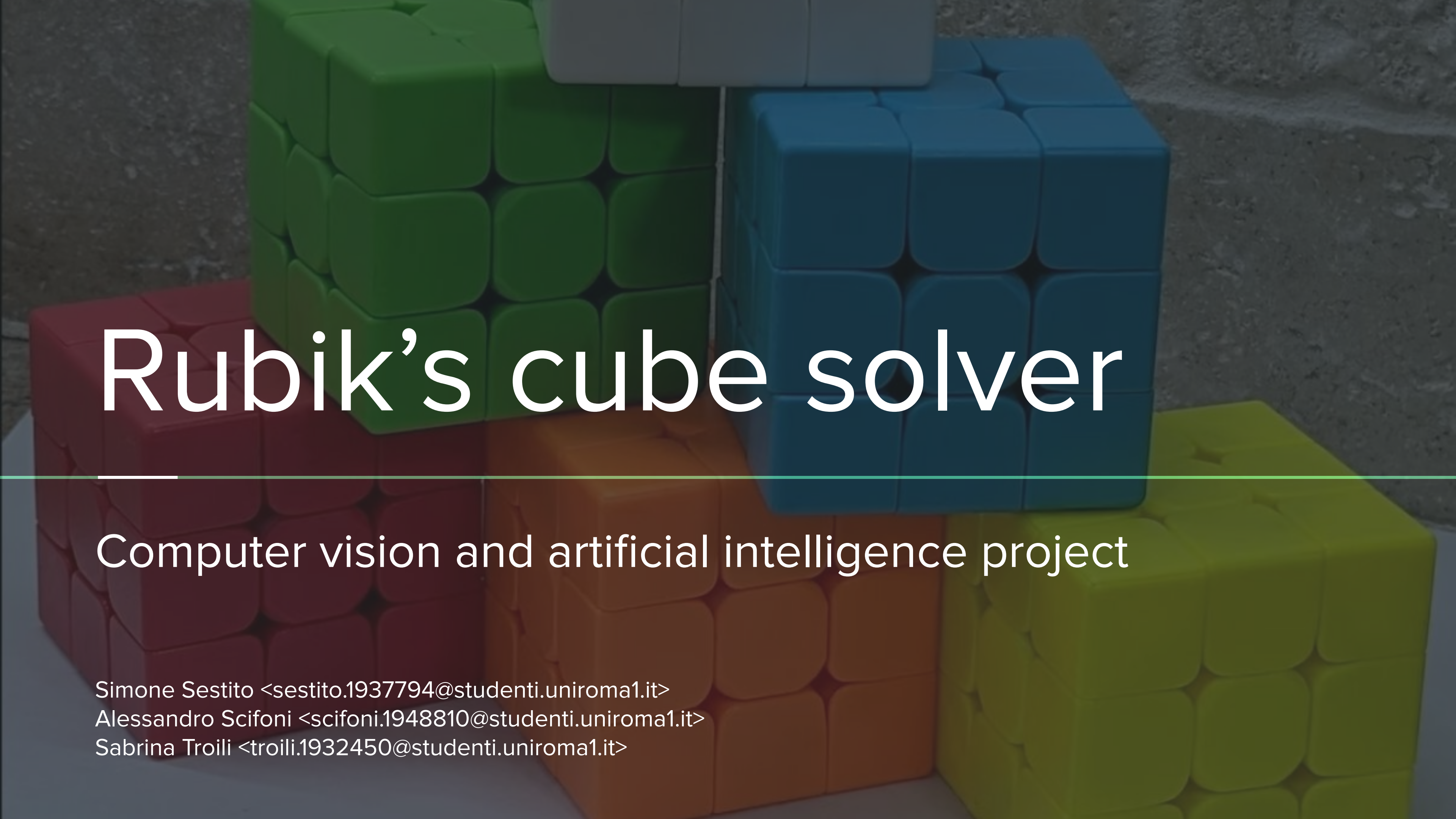


Rubik’s cube solver

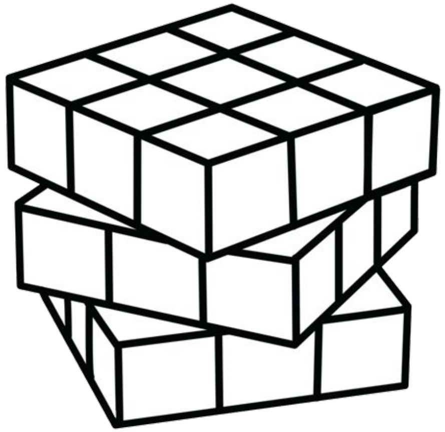
Computer vision and artificial intelligence project

Simone Sestito <sestito.1937794@studenti.uniroma1.it>
Alessandro Scifoni <scifoni.1948810@studenti.uniroma1.it>
Sabrina Troili <troili.1932450@studenti.uniroma1.it>

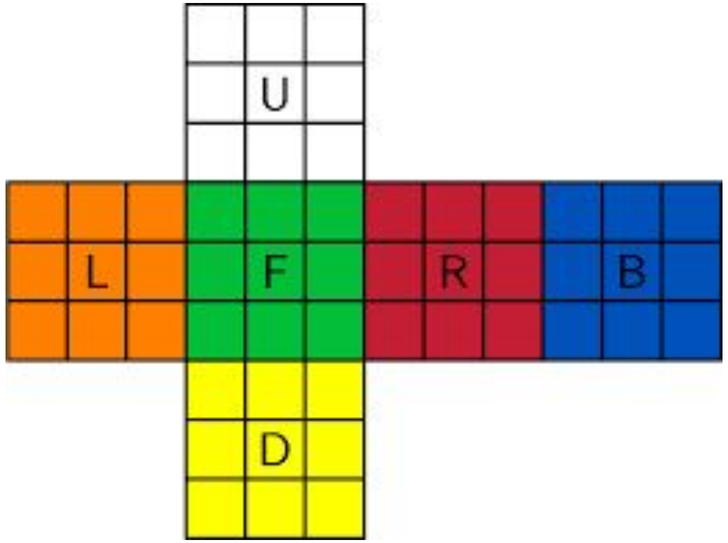
Project areas

Real-world cubes recognition

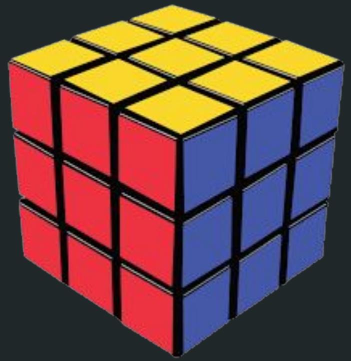
AI Move prediction



Cube Simulator

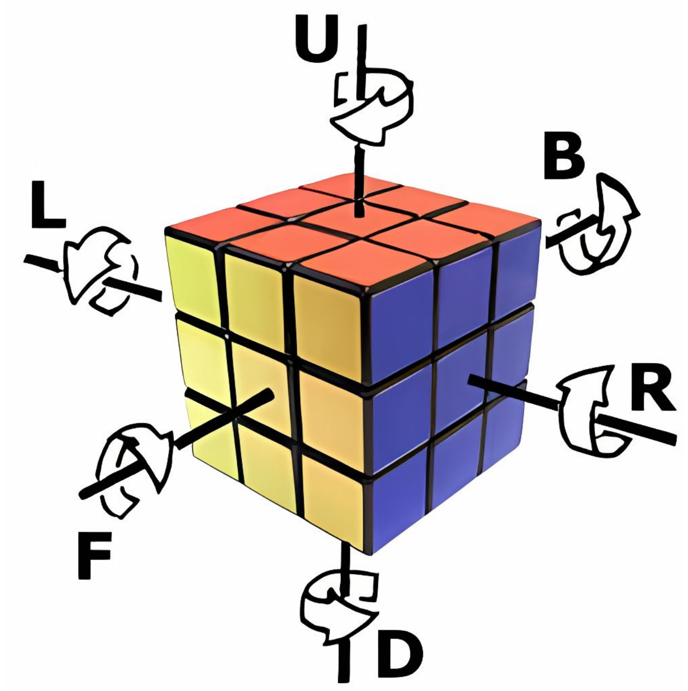


Flattened representation

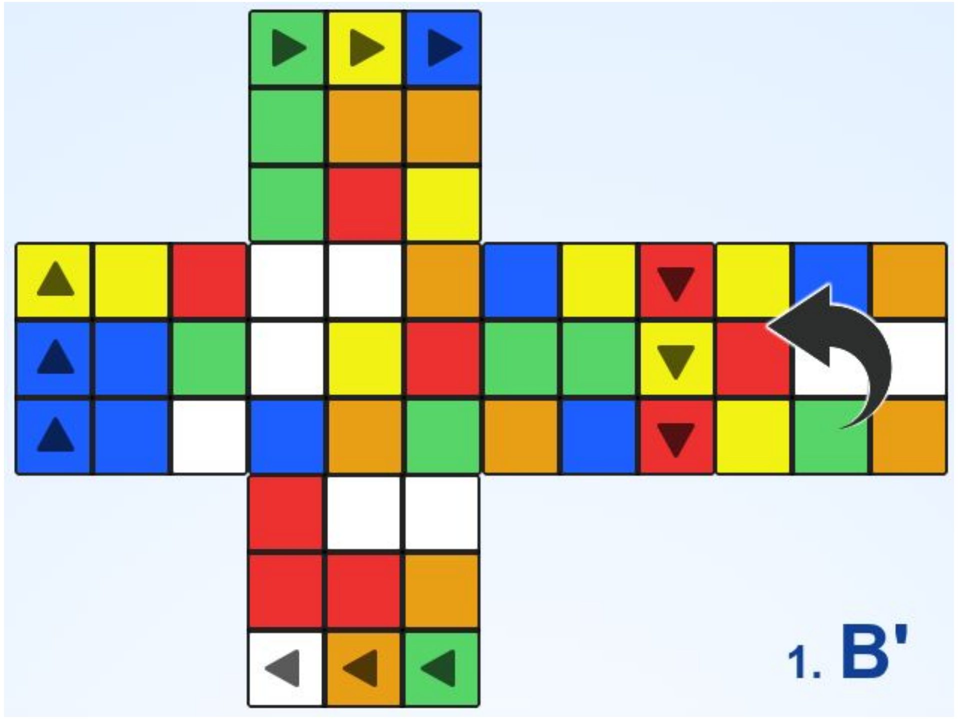


Standard representation

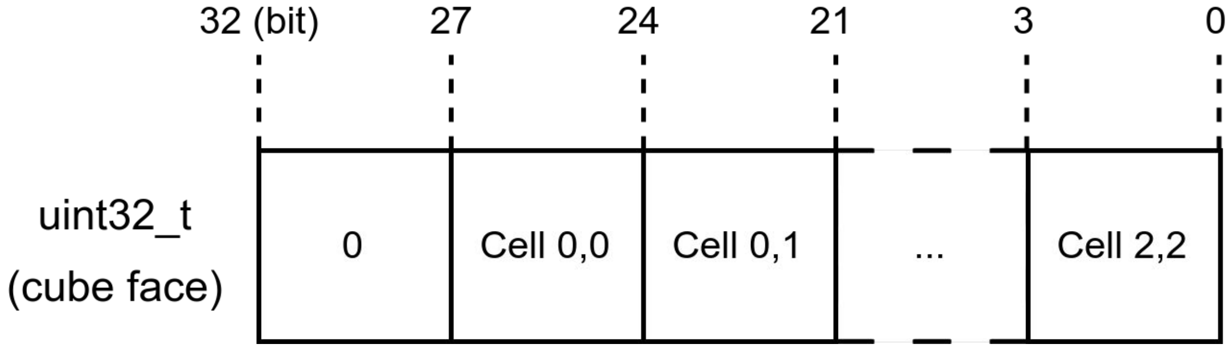
Standard notation



Rotate matrix and flip row or column?



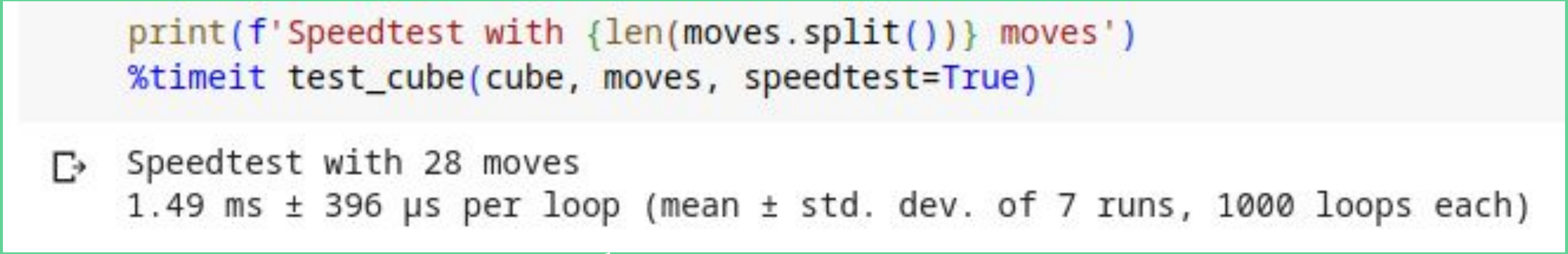
C optimization



Each cell can assume 6 values rappresentable with 3 bits,

so to represent a face we need 9x3=27 bits, rounded to 32 bits.

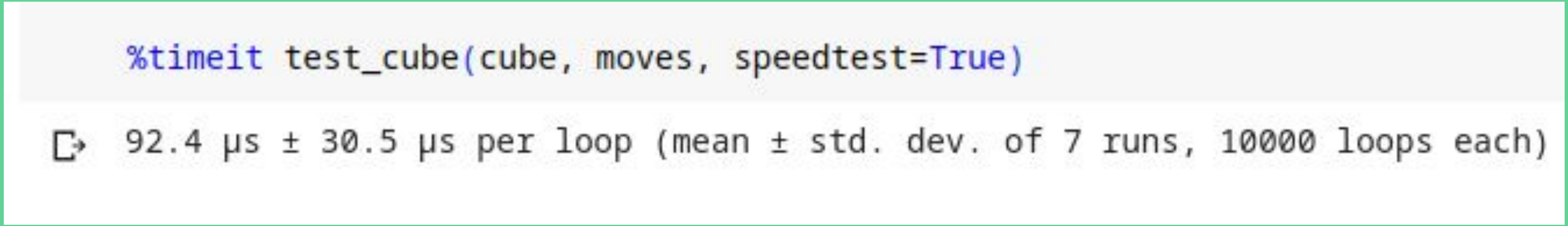
So a cube can be represented with 24 bytes!



```
print(f'Speedtest with {len(moves.split())} moves')
%timeit test_cube(cube, moves, speedtest=True)
```

```
Speedtest with 28 moves
1.49 ms ± 396 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

C implementation



```
%timeit test_cube(cube, moves, speedtest=True)
```

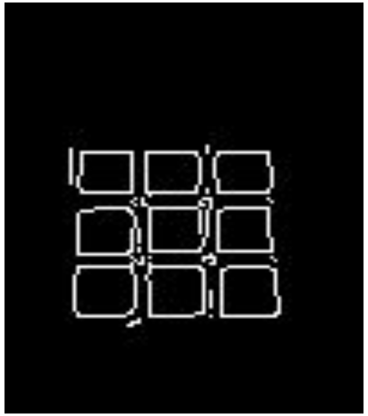
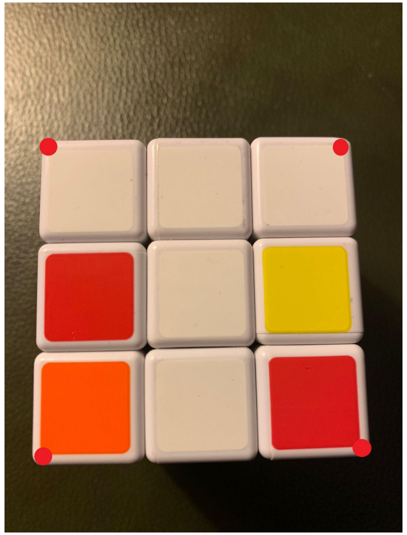
```
92.4 µs ± 30.5 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

OpenCV Recognizer

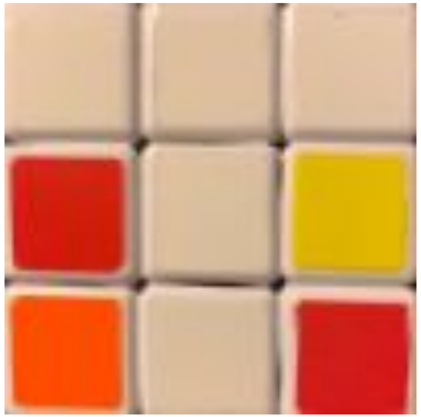
How to get the matrix?

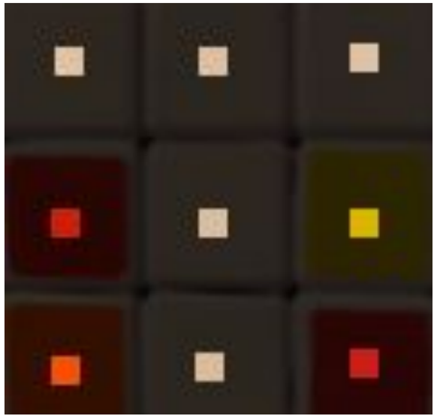
What are the steps?

1. Take pictures of the cube faces
2. Identification of the edges
3. Get the vertices

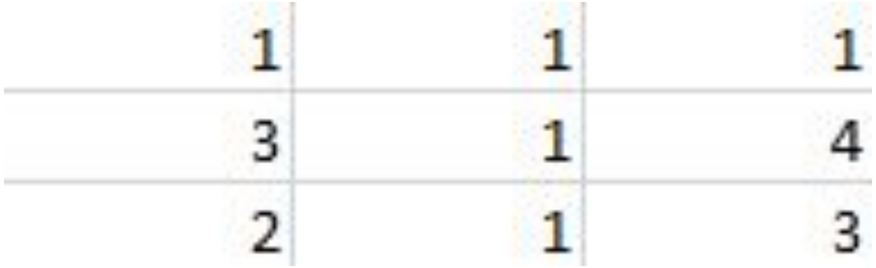
4. Eliminate the outer outline





5. Get a pixel from the cells

6. Build the matrix

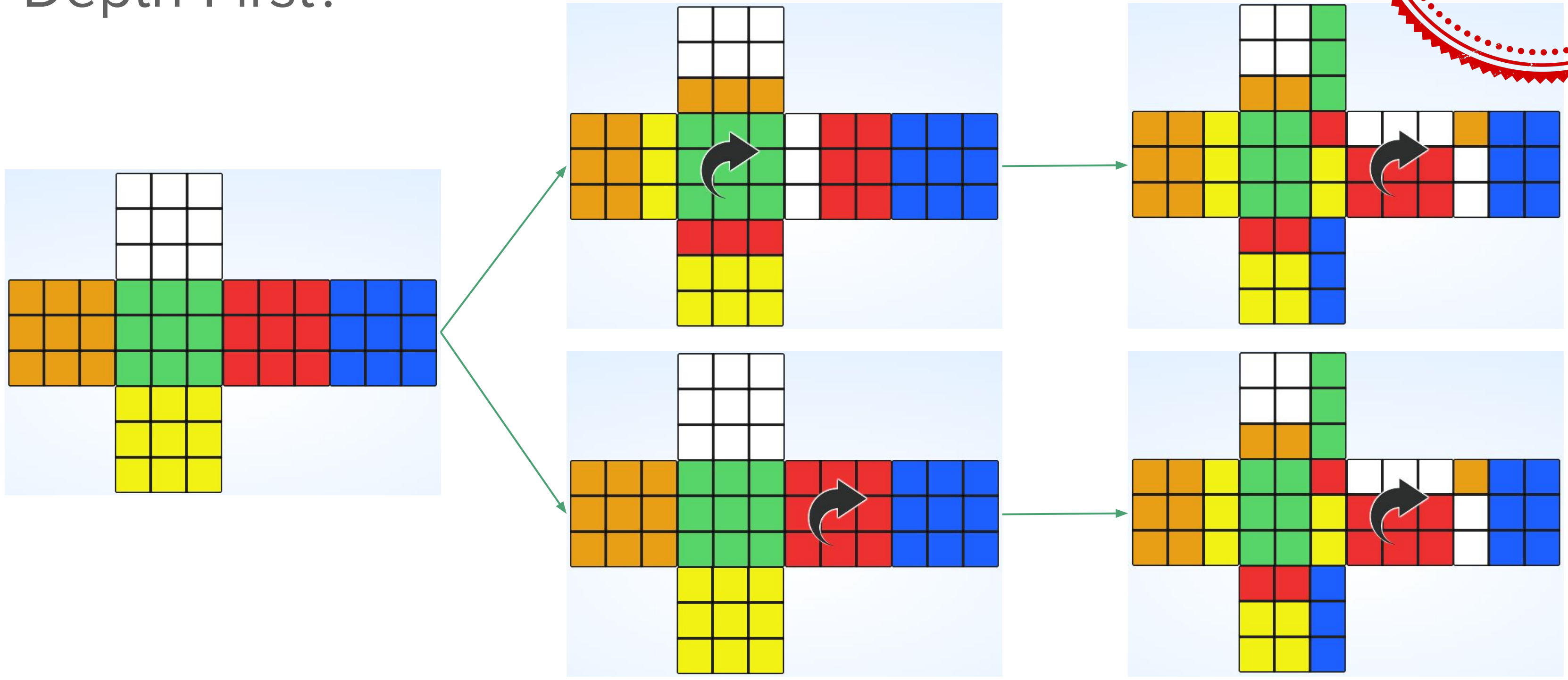


1	1	1
3	1	4
2	1	3

Dataset

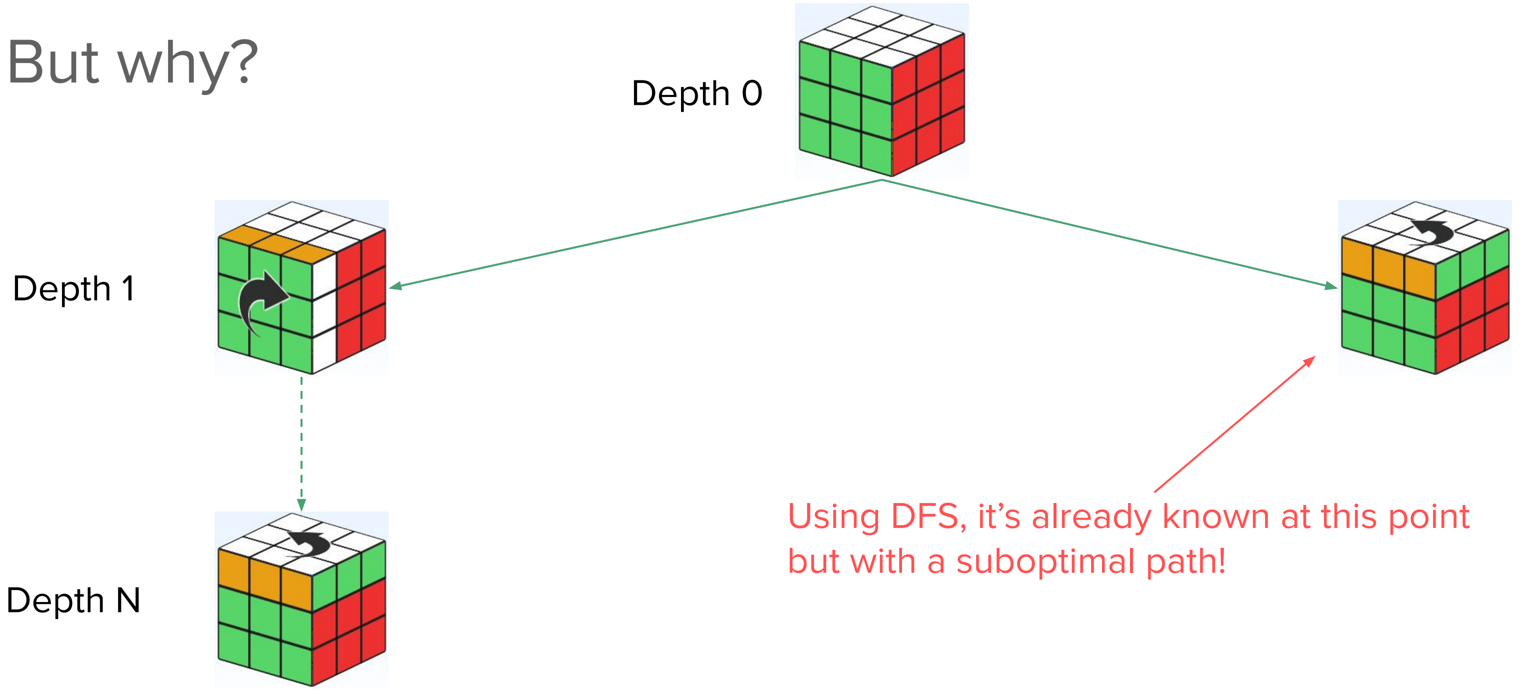
Dataset: how to find cubes

Depth-First?



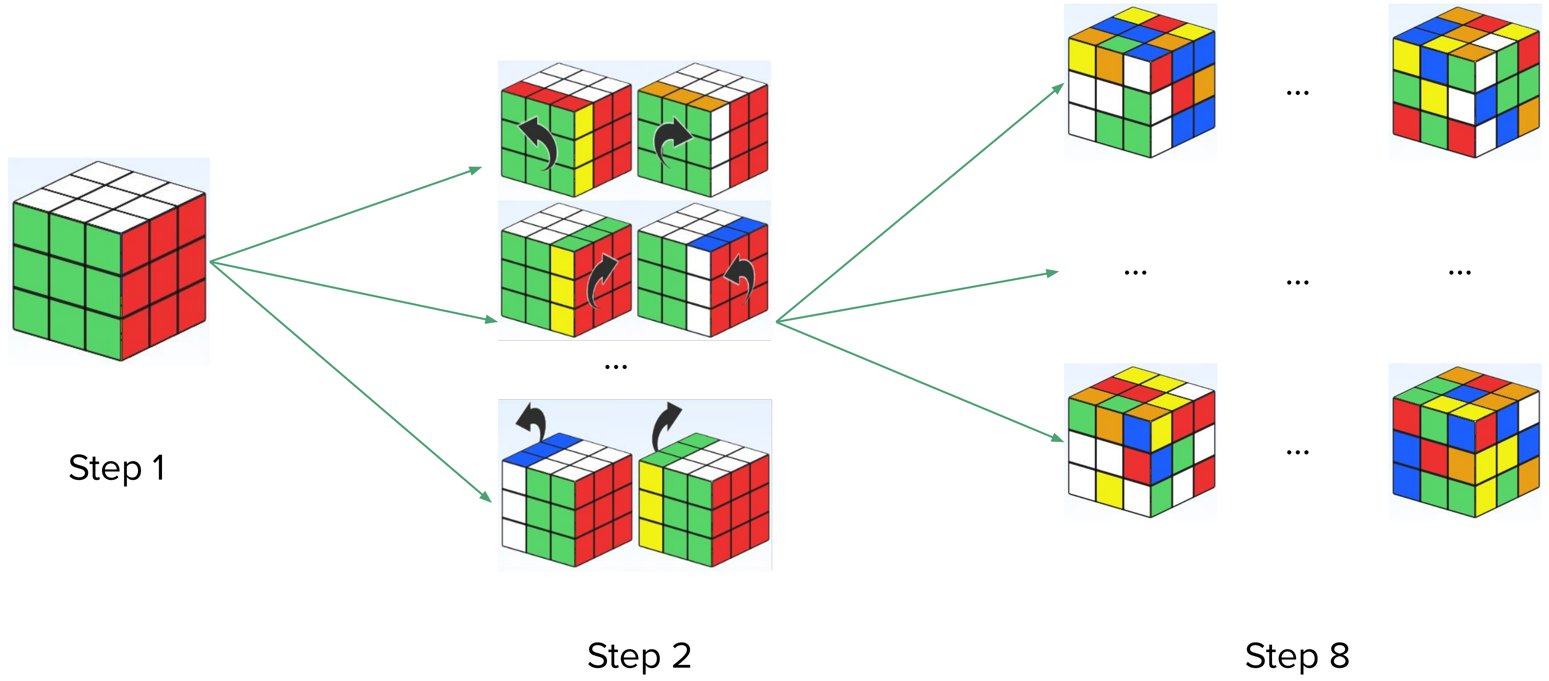
Dataset: don’t use DFS

But why?


Depth 0
Depth 1
Depth N
Using DFS, it’s already known at this point but with a suboptimal path!

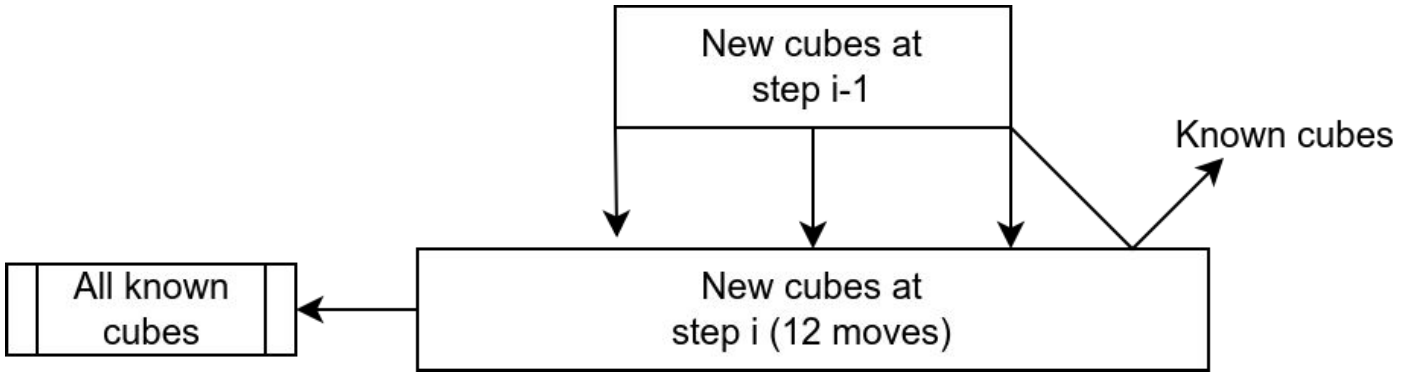
Dataset: how to find cubes, the right way

Breadth-First!



Dataset: how to find cubes, the right way

Breadth-First!

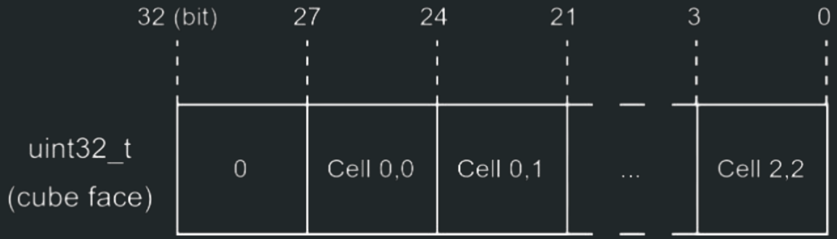


In a nutshell

3x3 Cubes Dataset

Used as a solver map for cubes recognized with OpenCV
≈ 86 million cubes
> 2GB on disk

We use the same optimized encoding of cubes in memory



for each cube face
+ 1 char for the move

2x2 Cubes Dataset

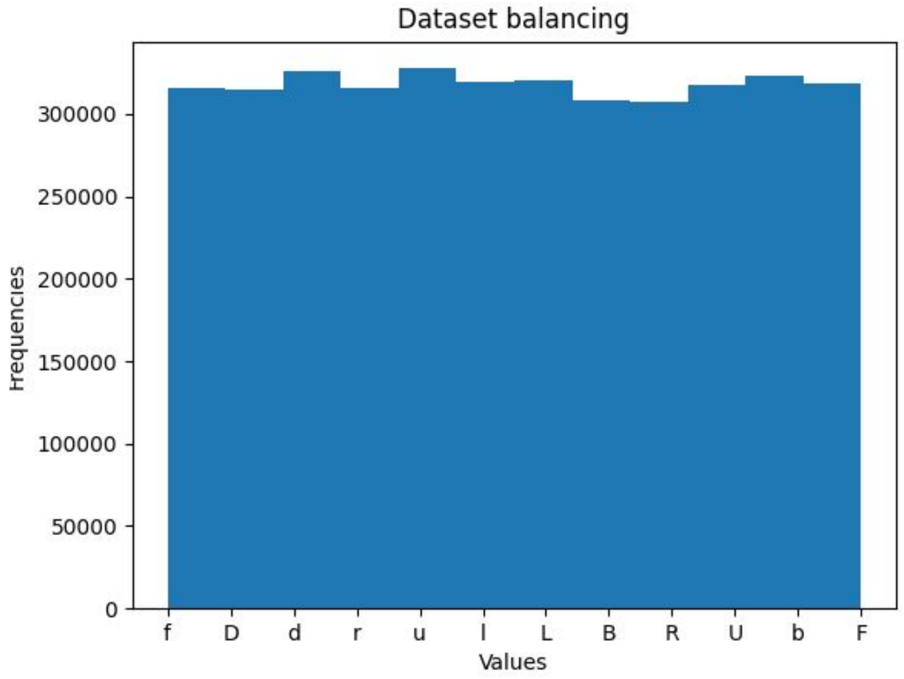
Used to train our AI model

Always with 8 moves
It’s just a ~250MB file!

No need to use an optimized encoding

Every cube is 24 Bytes + 1 char (move)

And it’s balanced!

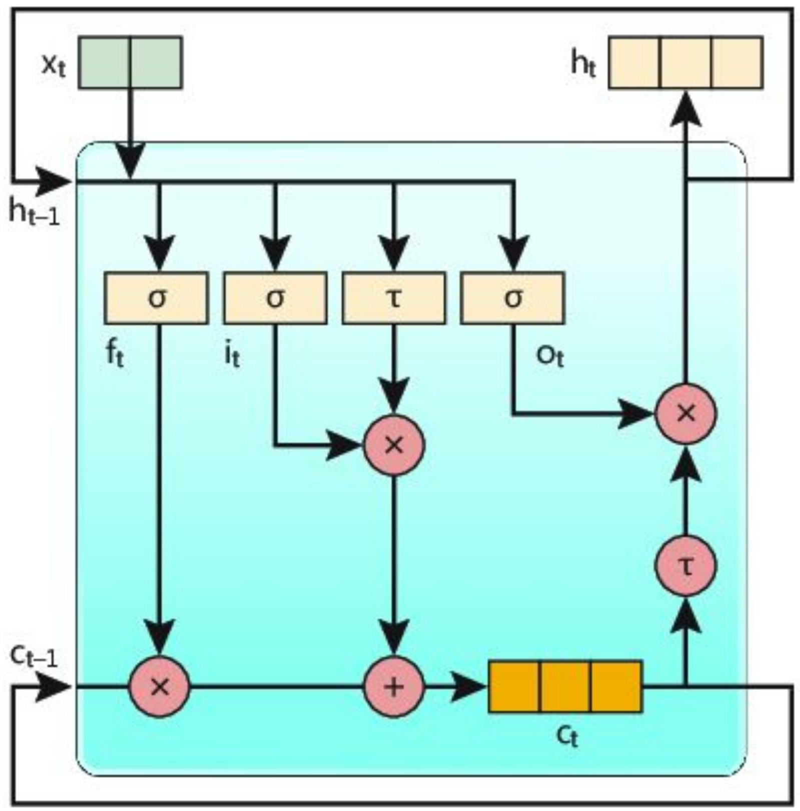


Samples per class, where the class is the optimal move to perform

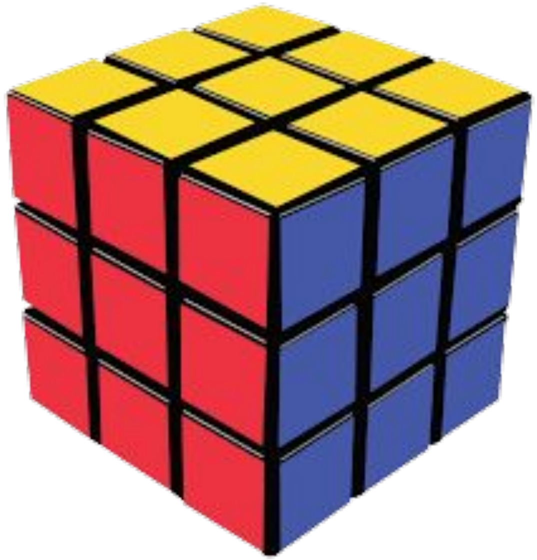
AI Model and Evaluation

LSTM Model

A model with memory.

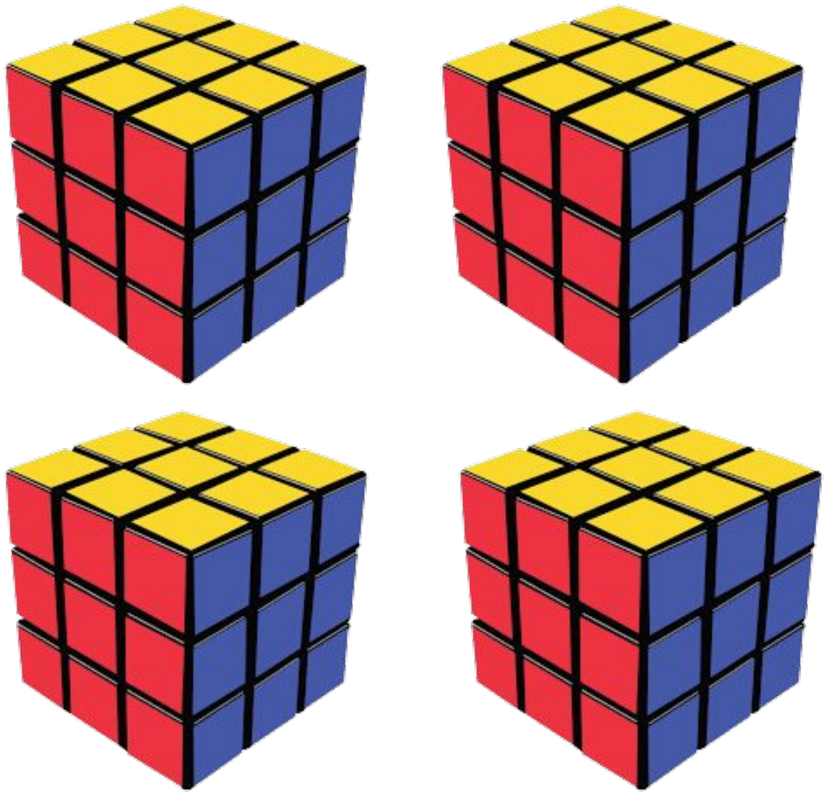


Dataset change: before

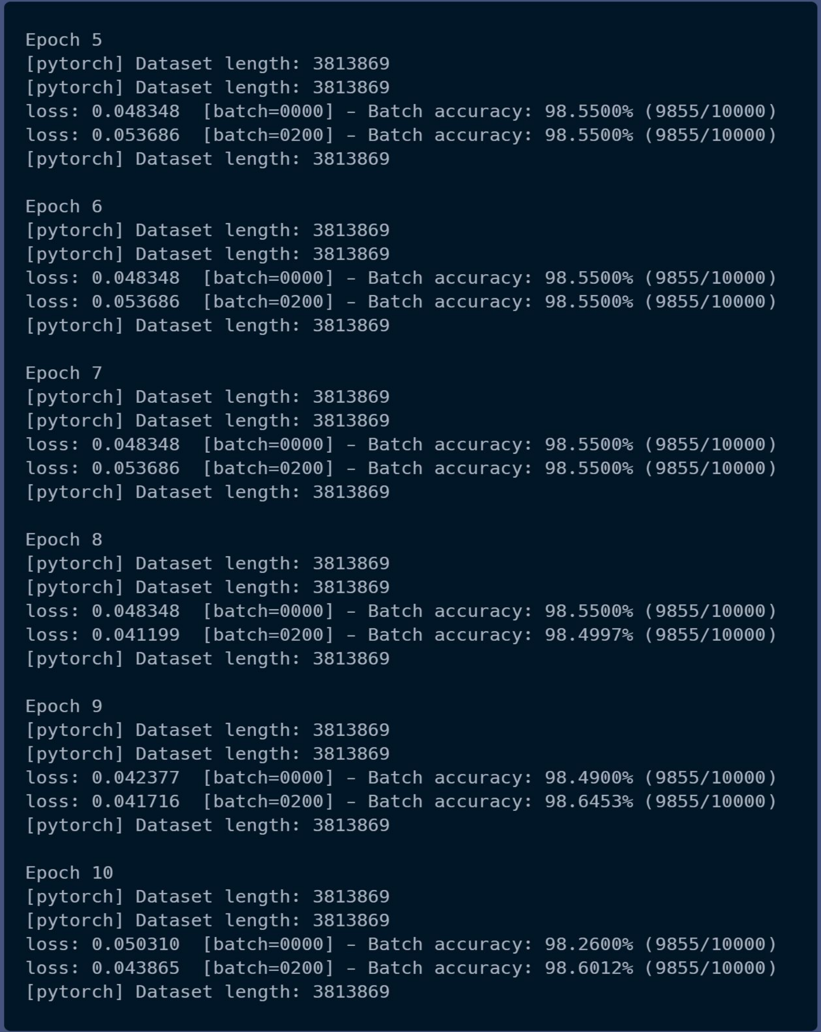


+ R

...AFTER



+ R



```
Epoch 5
[pytorch] Dataset length: 3813869
[pytorch] Dataset length: 3813869
loss: 0.048348  [batch=0000] - Batch accuracy: 98.5500% (9855/10000)
loss: 0.053686  [batch=0200] - Batch accuracy: 98.5500% (9855/10000)
[pytorch] Dataset length: 3813869

Epoch 6
[pytorch] Dataset length: 3813869
[pytorch] Dataset length: 3813869
loss: 0.048348  [batch=0000] - Batch accuracy: 98.5500% (9855/10000)
loss: 0.053686  [batch=0200] - Batch accuracy: 98.5500% (9855/10000)
[pytorch] Dataset length: 3813869

Epoch 7
[pytorch] Dataset length: 3813869
[pytorch] Dataset length: 3813869
loss: 0.048348  [batch=0000] - Batch accuracy: 98.5500% (9855/10000)
loss: 0.053686  [batch=0200] - Batch accuracy: 98.5500% (9855/10000)
[pytorch] Dataset length: 3813869

Epoch 8
[pytorch] Dataset length: 3813869
[pytorch] Dataset length: 3813869
loss: 0.048348  [batch=0000] - Batch accuracy: 98.5500% (9855/10000)
loss: 0.041199  [batch=0200] - Batch accuracy: 98.4997% (9855/10000)
[pytorch] Dataset length: 3813869

Epoch 9
[pytorch] Dataset length: 3813869
[pytorch] Dataset length: 3813869
loss: 0.042377  [batch=0000] - Batch accuracy: 98.4900% (9855/10000)
loss: 0.041716  [batch=0200] - Batch accuracy: 98.6453% (9855/10000)
[pytorch] Dataset length: 3813869

Epoch 10
[pytorch] Dataset length: 3813869
[pytorch] Dataset length: 3813869
loss: 0.050310  [batch=0000] - Batch accuracy: 98.2600% (9855/10000)
loss: 0.043865  [batch=0200] - Batch accuracy: 98.6012% (9855/10000)
[pytorch] Dataset length: 3813869
```

FIRST EXPERIMENT:

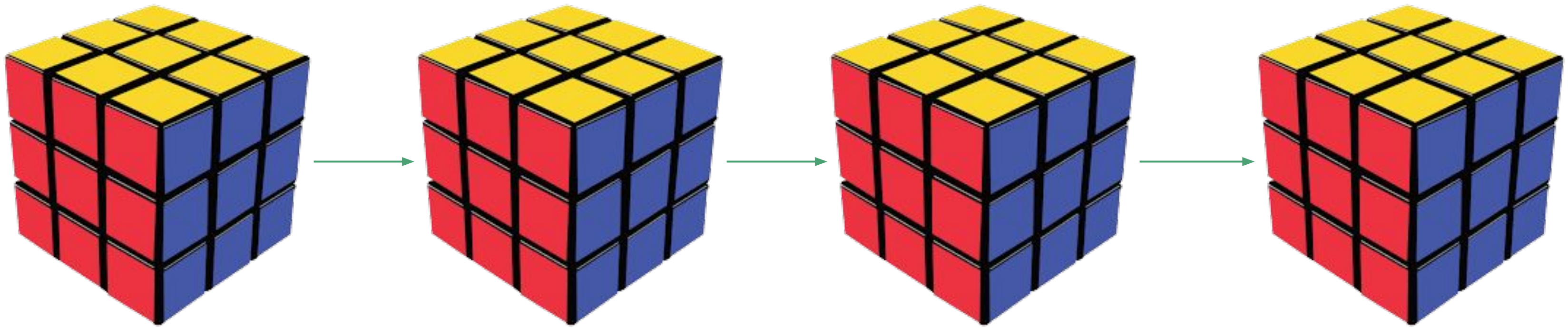
Series of tests using 10 epochs. Observation that starting with the 5th epoch, the results obtained are the same.

58%

9 moves

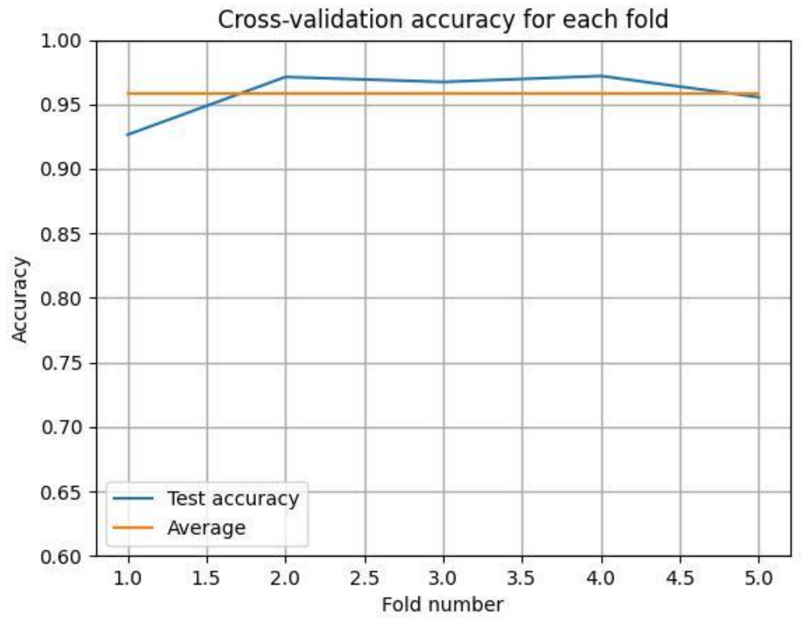
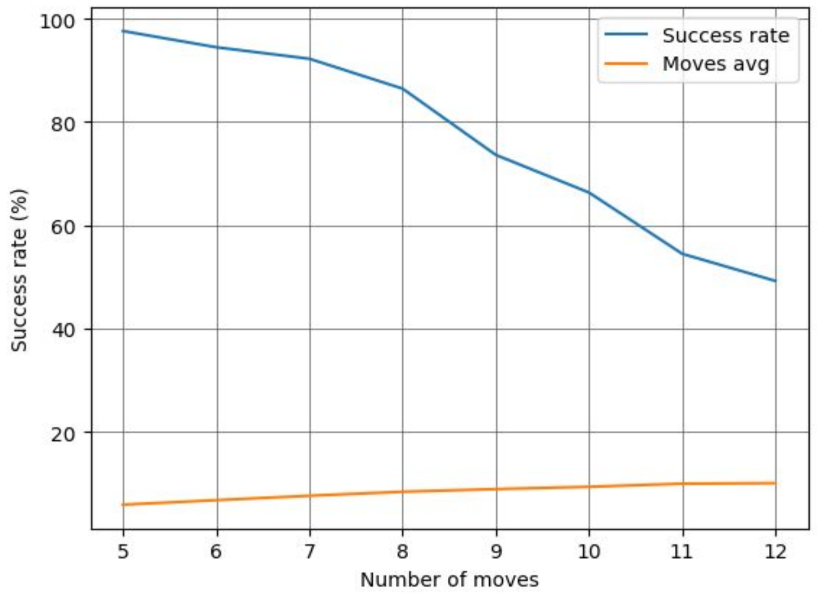
98%

8 moves



65%

98%


Cross-validation accuracy for each fold
Accuracy
Fold number
Test accuracy
Average
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

Success rate (%)
Number of moves
Success rate
Moves avg
100
80
60
40
20
5
6
7
8
9
10
11
12

Demo time!