Rubik's Cube Solver

CS156 - Machine Learning for Science and Profit
Prof. Sterne

CLASS CODE: 2023
Minerva University

Kalyane de Oliveira Bezerra
December 2021

Problem Definition **2**
Solution Specification **2**
Testing and Analysis **4**
References **8**
Appendices **8**

1. Problem Definition

There are 43 quintillions of possible combinations of a Rubik's cube. If you could do one movement every second, you would need more than the universe's time to get to all of these combinations. The universe has 14 billion years. There are many algorithms that people use to solve a cube, and for this reason, the Mundial record is 3.47 seconds. Exhaustive searches have shown that any cube can be solved in at most 26 movements (Irpan, 2016). But to get to this rapid solution, people need to train for many hours. As you are trying to learn how to solve the cube, it becomes difficult for you to transform a solved cube into a different valid combination, which could be helpful for you to create a Rubik's cube mosaic, for example.

As we already know, there are only 6 colors and 9 stickers each in a Rubik's cube, and for that reason, we can't express a cube in a 3x3x3 matrix as each cubie can have a different orientation. In addition, each cube can turn 90 degrees in either direction, totalizing 12 possible movements. If we could express a cube with a dimension for each cubie, there wouldn't be a clear linear decision boundary that would help us classify the cube. In this case, we have a non-linear problem, and Neural Networks can perform well on this problem because they can map the input data to a higher-dimensional space, and it has different non-linear activation functions that use weights and biases learned during training, so can learn the features of a Rubik's cube. I believe Neural networks are the best algorithm learned in CS156 to solve this problem. We can do hyperparameter tunings such as epoch, training rate, and loss function that help achieve better accuracy when using lots of training data.

2. Solution Specification

The idea presented in this assignment is that creating a neural network that outputs the depth of a Rubik's cube would be enough to transform a combination into a solved state or vice versa (Johnson, 2021). One constraint is that if we consider that any cube can be solved with a max of 26 movements, we would need to have much more data. I found a solution to assume that a cube has a maximum of 9 moves to generate a dataset with linear growth of the amount of data for each depth to avoid a huge imbalanced dataset (the Rubik's cube is an exponential problem). The dataset contains 4.5 million cubes with 100.000*depth random cubes for each depth.

The Rubik's cube is represented by a 2D matrix of 6x9 (colors x stickers), and for each movement function, I cautiously manipulated the swap of colors to avoid any invalid combination. There is also a visualization function to see the cube in a 2D format, and it is helpful to a human to follow the steps.

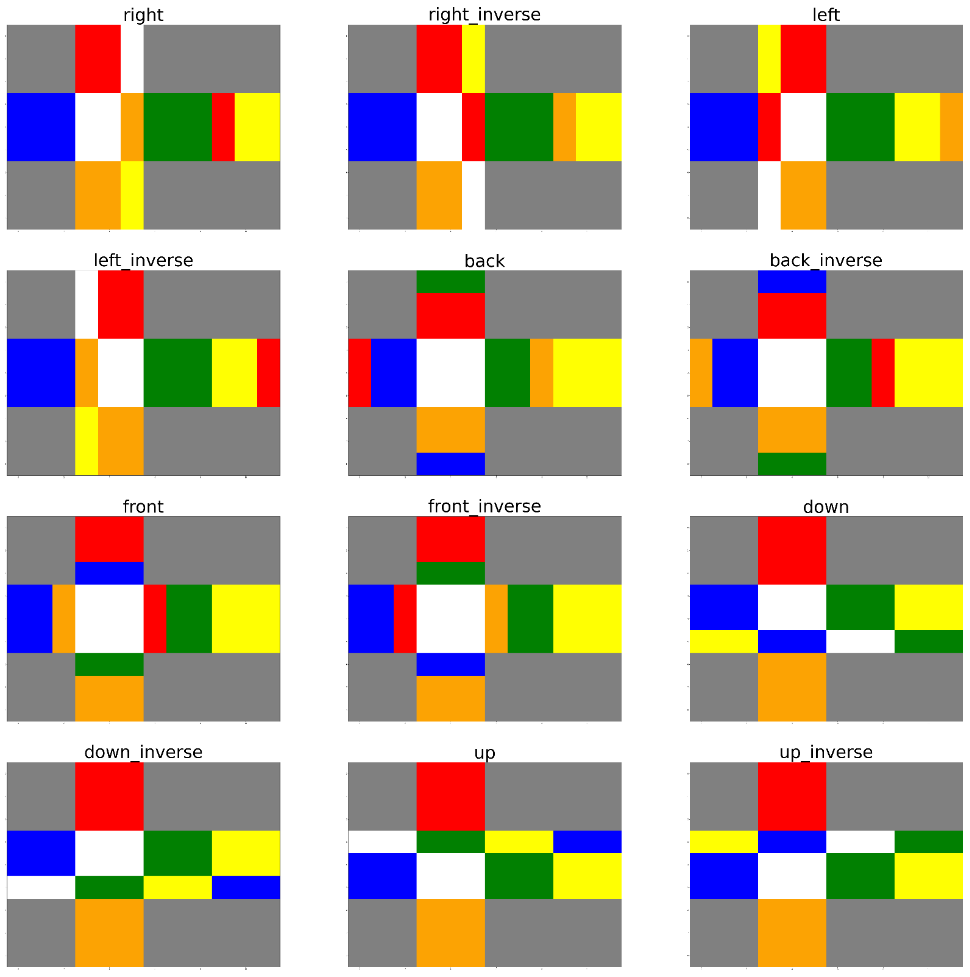


Figure 1: All 12 valid Rubik's cube movements.

To create the Neural Network, I had an idea for the input (54 neurons representing each sticker) and output (0-8 softmax depth) layers but had no idea of the best hyperparameters. For this reason, I decided to use Keras Tuner to choose the number of hidden layers, activation function, number of neurons, and learning rate. To compile, I used the Sparse Categorical Crossentropy as the loss function. It would have a higher loss for predictions away from the actual value and the accuracy for the metrics because it can show how close a measurement comes to its true value. To avoid losing the hard work, I saved all trials and selected the best. I initially used 100 epochs and default steps to train the model, but I had a memory error and continued with 50 epochs and 45000 steps. As I was using a stopping early callback, it stopped training when the loss wasn't changing for the last 5 epochs. In the end, I achieved ~40% accuracy.

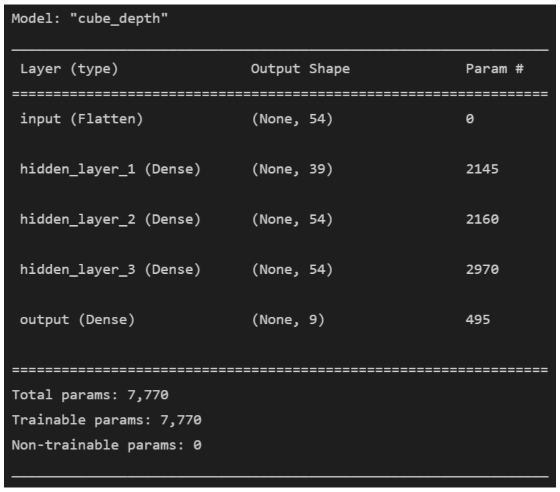
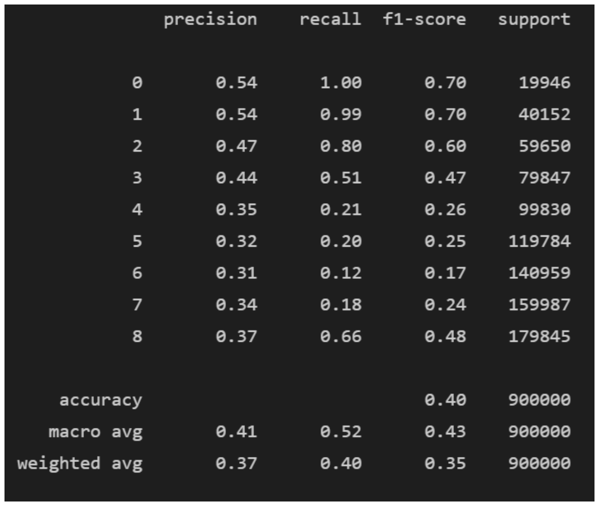

```
Model: "cube_depth"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (Flatten)             (None, 54)                0

 hidden_layer_1 (Dense)      (None, 39)                2145

 hidden_layer_2 (Dense)      (None, 54)                2160

 hidden_layer_3 (Dense)      (None, 54)                2970

 output (Dense)              (None, 9)                 495

=================================================================
Total params: 7,770
Trainable params: 7,770
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Summary of the generated model using Keras Tuner.

3. Testing and Analysis

As I had previously divided the dataset into training (0.8) and testing (0.2), I could try to predict the depth of each cube and compare it with the actual values. By doing that, I created a classification report and a confusion matrix. From the classification report, we can see that the precision (accuracy of positive predictions) and recall (positives that were correctly classified) decreases when the depth increases but with a slightly larger value for the biggest depths that might be a result of having much more cubes with depth 9 than with depth 6. From the confusion matrix, we can see that the incorrectly classified values were close enough to the actual value, for example, classifying 7 when the real is 6. This is good because even if the classification is incorrect, it is still meaningful, and we have more chances of finding the correct movements.



```
              precision    recall  f1-score   support

           0       0.54      1.00      0.70     19946
           1       0.54      0.99      0.70     40152
           2       0.47      0.80      0.60     59650
           3       0.44      0.51      0.47     79847
           4       0.35      0.21      0.26     99830
           5       0.32      0.20      0.25    119784
           6       0.31      0.12      0.17    140959
           7       0.34      0.18      0.24    159987
           8       0.37      0.66      0.48    179845

    accuracy                           0.40    900000
   macro avg       0.41      0.52      0.43    900000
weighted avg       0.37      0.40      0.35    900000
```

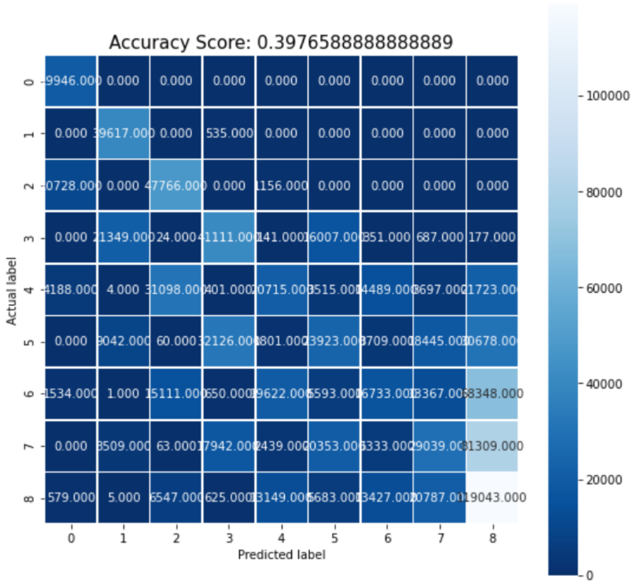


Figure 3: Classification report and Confusion matrix on the testing dataset.

To test the general idea of my algorithm, I created a function that gets the depth of the cube and another that tried at most 12 movements to solve the cube. To find the proper movement, I take the current cube, turn it with all possible moves, and save the move with the minimum depth. This way, it creates a sequence of steps that a human can perform from the scrambled Rubik's cube to the solved state. Going from the solved state to the scrambled just requires you to inverse the list. After doing this, I created 100 new test samples for each depth and recorded if it was solved and how many steps it took to solve. Doing this, I could generate plots about its solvability and number of movements. From the solvability graph, we can see that until the 4th depth, it has mostly solvable cubes, but there is a linear growth of unsolvable cubes in each larger depth. From the graph of movements, we can see that we had cases that the algorithm could solve with 1 step in almost all depths. This happens because if we perform 2 movements like left and left_inverse it can also be considered as having done no moves as the second undid the first. Another interesting piece of information from the graph is that on the 6th and 7th depth it had a cube that the algorithm solved with more steps than the actual depth, which shows that we can solve a combination in many different ways. But I am glad the other depths had the number of movements equal to the depth or less, showing that my Neural Network helps me find the minimum number of moves to solve.

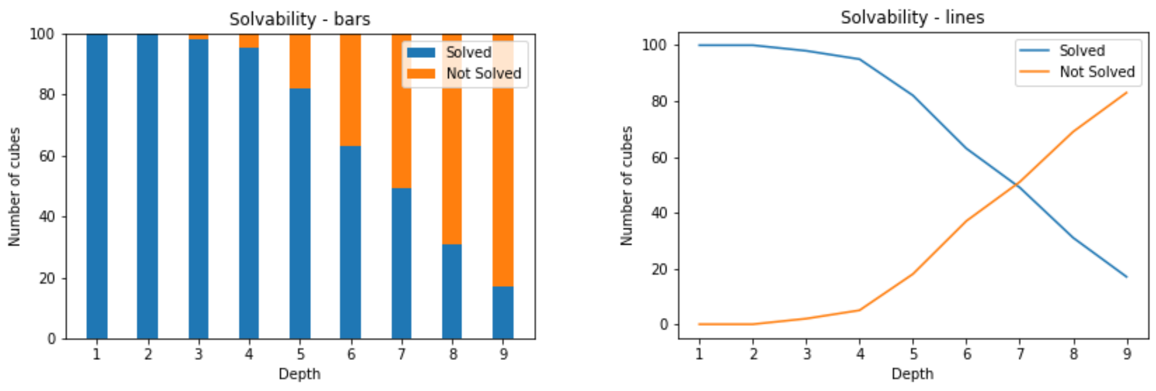


Figure 4: Graphs showing the number of random cubes that were and weren't solved.

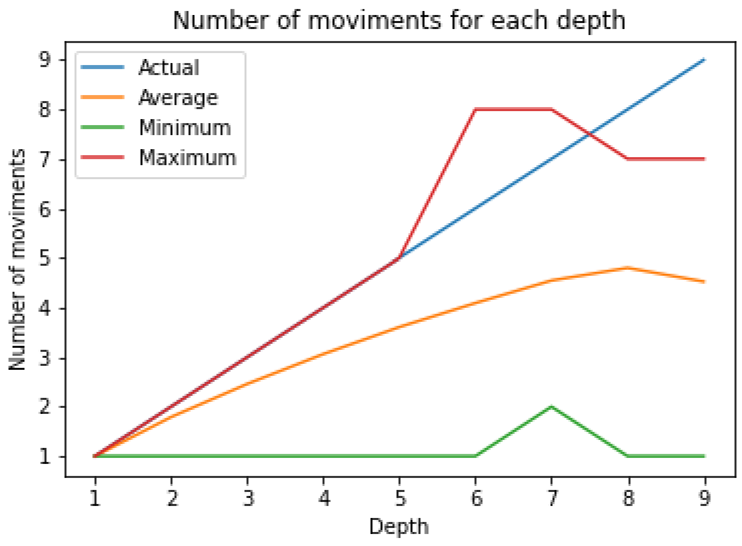


Figure 5: Graph showing the number of movements for each depth.

As the initial idea of my final project was to create a mosaic out of Rubik's cubes, I decided to illustrate one example of a 3x3 matrix of pixels of a Mario mosaic if we knew the final cube configuration. I just couldn't generate a valid cube out of a 3x3 matrix of pixels because the Rubik’s cube rules are too complicated (), and I don’t know how to create a Neural Network that could have an output layer with a classification of color for each neuron.

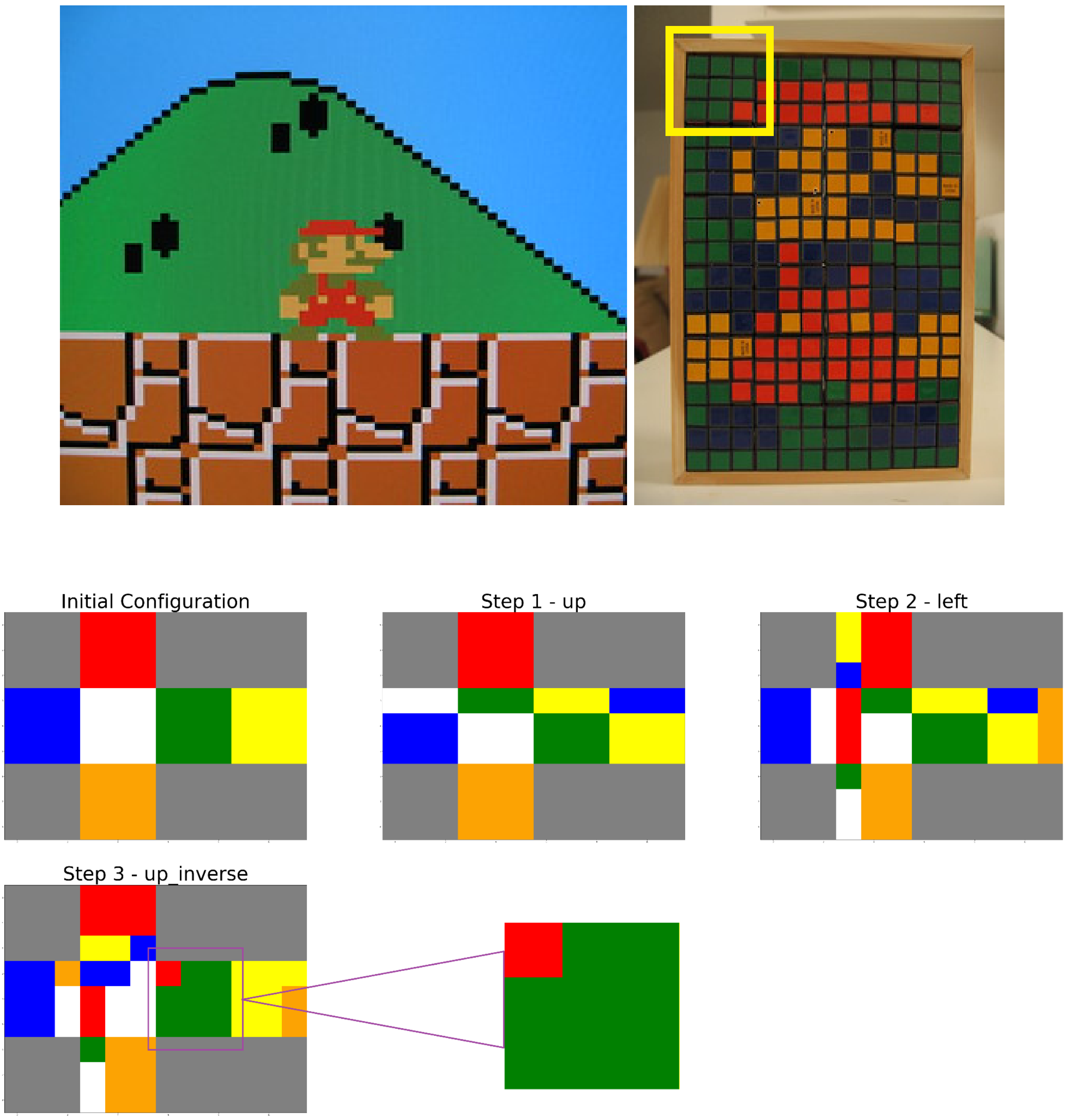


Figure 6: Example of creating a simple mosaic piece.

4. References

Chen, Janet. (2004). Harvard. *Group Theory and the Rubik's Cube.* Retrieved December 26, 2021, from Group Theory and the Rubik's Cube

Irpan, Alexander. (2016). h Conference on Neural Information Processing Systems. *Exploring Boosted Neural Nets for Rubik's Cube Solving*.

Johnson, C. G. (2021). Solving the Rubik's cube with stepwise deep learning. *Expert Systems*, *38*(3). https://doi.org/10.1111/exsy.12665

Team, K. (n.d.). *Keras Documentation: Kerastuner*. Keras. Retrieved December 25, 2021, from https://keras.io/keras_tuner/

5. Appendices

Unfortunately, I was not able to create the full mosaic using my neural network, but here is the link to my GitHub repository where you can find all the code to generate the dataset (it was too large to push), create the model, and testing:
https://github.com/kalyane/rubik-s-cube-mosaic-creator

It also includes my failed attempt of generating a Rubik's cube out of a 3x3 matrix and trying to validate by web scrapping a website.