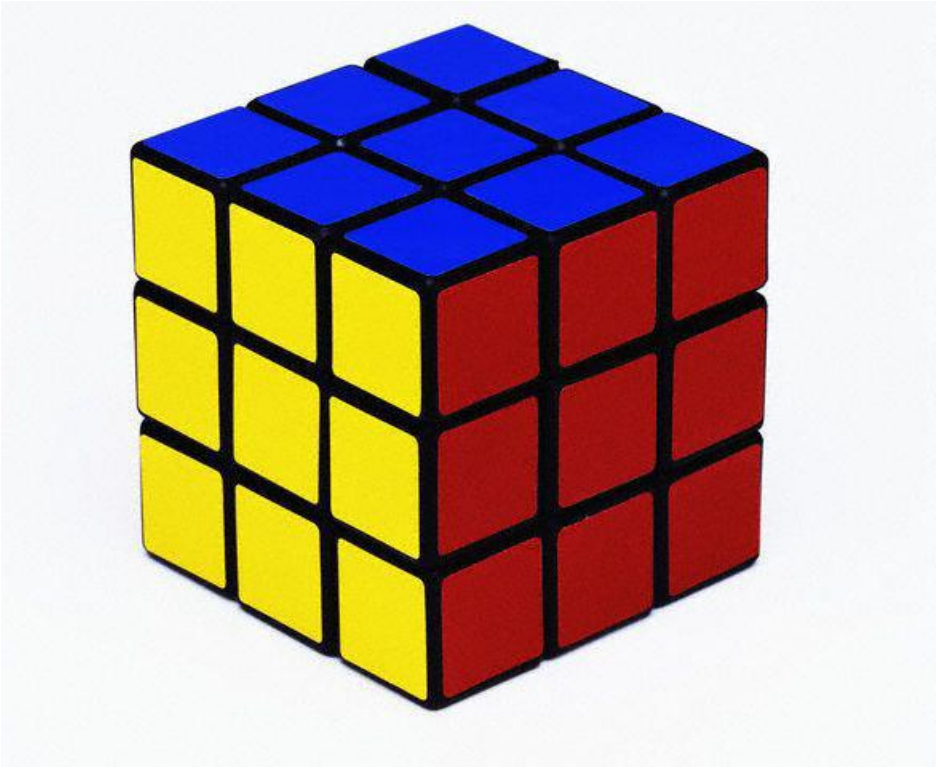


Gesture Controlled Rubik Cube Android Application

ECE 1780 PROJECT REPORT

xiaoyu sun | 1004640230 | April 5, 2018

Introduction

Rubik's cube is a very classic puzzle game[1][2]. With smartphones become popular, there are lots of different versions of Rubik's cube game application published on both Android and IOS platforms. In recent years, with the publication of deep learning interface and TensorFlow Object Detection API, people start to think of other ways to implement their functions. Under this background, this project implemented a gesture-controlled Rubik's cube game using real-time object detection through the camera. This is an example of a new way to play such classic games. Furthermore, as the operations of Rubik's cube game are very basic but diverse, the operation part of this application can also be used to many other applications.

In this project, a simplified version of gesture-controlled Rubik's Cube application is implemented based on TensorFlow Object Detection API[3]. In this application, players can control the cube with the movement of fingers, which is similar to reality. However, limited by the detection model, the gestures that can be well recognized are not closed to the gesture we used in reality to control the cube. This will be optimized in future works.

This report is organized as follow: (1) the first part introduced the main objectives of this project; (2) based on the objectives, this part included the detailed solution and implementation of the application designed in this project; (3) the demonstration part displayed the screenshot of the demo of this application; (4) this part introduced several way to improve the application and some extension of the method being used.

Objectives

1. Learning Model:

   As a mobile game application, the calculating speed of the machine learning model will significantly influence the user experience. So, model choosing and optimizing will be the two main problems in this project.

   Another element which will also influence the application performance is the detection accuracy. In this application, the size of the dataset and the number of training iterations is the two features that decide the detection accuracy.

2. Android Application:

   For application design part, the most important part will be the game logic design. There are mainly three problems included. The first one is the game logic design, which includes basic game logic implementation, and game strategy design. In this application, the game logic will simply use the original logic of the Rubik's Cube. But for the game strategy, the most popular twelve option strategy is too complex for controlling with gestures. For this reason, the strategy must be simplified.

   The user interface design is also very important. Rubik's cube is originally a 3D puzzle game. In this application, the 3D cube will be displayed in a 2-dimensional way. If the interface is not carefully designed, it will be hard to play even for a professional Rubik's cube player. In one word, user interface design decides the user experience directly.

Detailed Design and Implementation

1. Learning Model:

   As mentioned, the learning model is based on the TensorFlow Object Detection API. The training process followed the tutorial published by Daniel S.[4].

   Model selection: As mentioned before, the model used for this application should compute fast. To achieve this, the SSD Mobilenet model published by TensorFlow Object

Detection API is selected as the detection model in this project. The attributes for other available models are listed in Appendix A. Table 1 shows the detailed hyperparameter settings for SSD Mobilenet Model.

Table 1: Basic Settings of Mobilenet Hyperparameters

Parameter	**Class Num**	**Matched threshold**	**Layer Num**	**Data width**	**Data height**
Value	5	0.5	6	300	300
Parameter	**Dropout Pr**	**Activation**	**Regularize**	**Learn Rate**	**Optimizer**
Value	0.8	ReLU_6	l2 (MSE)	0.004	Momentum

Dataset collection: As mentioned above, the detection accuracy will be significantly influenced by the variance of the dataset. In this project, there are around 20 pictures captured from 5 students for each gesture. To simplify model training, the background of all these pictures are set to white. This setting will limit testing accuracy, since the model will only perform well with the white background.

2. Application User interface design

To make the application friendlier, the layout is designed to be a box-style expend of the top, front, bottom, left and right surface. After a survey of several students, the back surface uses a perspective illustration. The layout illustration of a level-2 Rubik's cube is shown in fig 1.

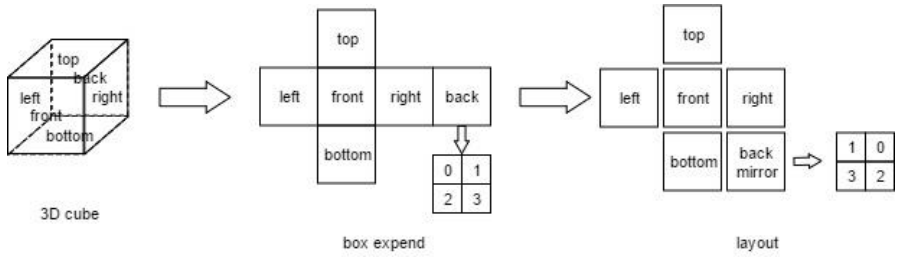


Figure 1: Level-2 Rubik's Cube Layout Illustration

3. Game design:

Based on the user interface introduce above, this part will mainly focus on the game logic design. In this application, the basic game logic implements the original logic of Rubik's cube game, but is a simplified version which only included 6 operations with 8 motions been realized.

Game strategy design: As introduced above, there are only 6 operations available which implemented 8 motions. The 6 operations include select, deselect, move up, move down, move left and move right, which refers to 5 gestures include fist-palm (select and deselect), up, palm (move down), left and right. The 8 motions implemented is shown in Table 2.

Table 2: Relationship between Operations and Motions

operation	**up**	**down**	**left**	**right**
Block selected	Column rotate up	Column rotate down	Row rotate left	Row rotate right
Block unselected	Cube rotate up	Cube rotate down	Cube rotate left	Cube rotate right

Basic Game logic implementation: In this game, the logic of cube rotation is easier to implement. For example, if the cube rotates up, the front, top, back and bottom surface will

change to top, back, bottom and front surface. At the same time, the right surface will perform a clockwise rotation, and left surface rotates to opposite direction.

The column and row rotation are similar to cube rotation, but only half of the blocks will be changed. Fig 2 shows an example of how each surface changes when performed a left column up rotation in a level-2 Rubik's Cube game.

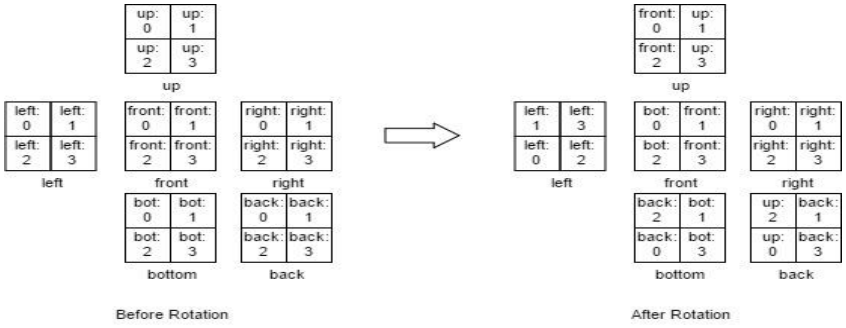
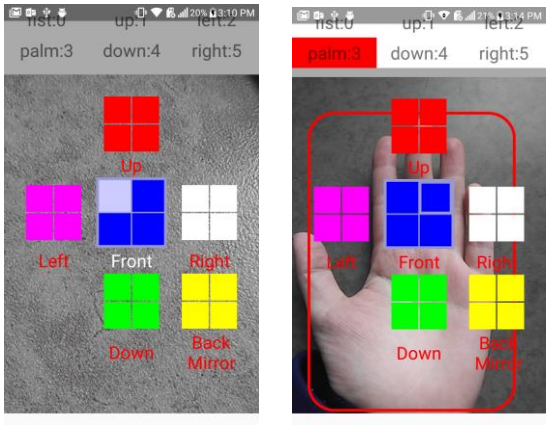


Figure 2: Block illustration before and after a left column up rotation

4. Links between three parts:

At the time this application start, the layout controlling thread will be initialized first. Then, the detection model object will be created and run in the background. When the model detected a gesture, the corresponding operation number will be returned to the game. After the game object calculated the new block colors map, the map will be transferred to layout controlling thread. Finally, after the layout thread accepted the new color map, the frame will be refreshed.

Demonstration



(a) (b)

Figure 3: Demonstration of Implemented GUI

Fig 3 shows the screenshots of the implemented application. There are mainly 3 parts in the frame: the operation list on top, the 2-level Rubik cube game in front, and the camera detection demonstration at back.

The operation list shows current operation status. For example, in Fig 3(a), the application just started with no operation detected. At this time, the current pointed block is the top left block in "Front" (shown with light blue). In Fig 3(b), Palm is detected, and the corresponding operation (shown as "palm:3" with red background) is recognized. Then, the block pointed currently (the top right block in "Front") is selected (reshaped to be smaller).

As the accuracy of the detection model is still not very satisfactory, the operation list is also made as operation buttons for testing and simulation.

Limitation and Future Work

To avoid recognition noises, there is a small gap set between each recognition. Even though, the accuracy is still limited by the background color. Another problem is the model may misrecognize up and left. To make the game flexible, the training data sampled both right and left hand, but the features of “left” gesture made by one hand is very similar to “up” gesture made by another hand. Since the data of left hand is more than right hand, the gesture detection accuracy of left hand is higher. Most of the problems listed here can be solved with a large training dataset and a deeper Neural Network in the future.

Another recommendation is for game interface expansion. Recently, there are already gesture controlled cameras. Inspired by this, the operation and reaction part can be reorganized as a portable package which can be used for other applications as well. The gestures included in this application already covered the basic operations for most mp3 players. With some further implementation of position detection, the operations will satisfy the controlling requirements of most applications.

Summary

This project implemented a gesture-controlled Rubik’s Cube game application of Android. Such design gives players more realistic experiences compared to traditional Rubik’s Cube application. The methods can also be extended and used to more other applications. But limited to the accuracy of the current model, the game experience is not very satisfactory. Better models and 3D game GUI is expected in the future.

Reference

[1] Fotheringham, W. (2006). *Fotheringham's Extraordinary Sporting Pastimes*. Robson.

[2] de Castella, Tom. *The people who are still addicted to the Rubik's Cube*. BBC News Magazine. BBC. Retrieved 28 April 2014.

[3] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., ... & Murphy, K. (2017, July). *Speed/accuracy trade-offs for modern convolutional object detectors*. In IEEE CVPR.

[4] Daniel S. (Oct 6, 2017). *Step by Step TensorFlow Object Detection API Tutorial*. Retrieved from https://medium.com/@WuStangDan/step-by-step-tensorflow-object-detection-api-tutorial-part-1-selecting-a-model-a02b6aabe39e

Appendix

Appendix A. attributes of different available object detection models. (resource from: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md)

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks
faster_rcnn_resnet101_kitti	79	87	Boxes
faster_rcnn_inception_resnet_v2_atrous_oid	727	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_oid	347		Boxes