Rubiks Cube Localization, Face Detection, and Interactive Solving

Jay Hack
Stanford University
Stanford, CA
jhack@stanford.edu

Kevin Shutzberg
Stanford University
Stanford, CA
jkevin@stanford.edu

Abstract

While Rubiks cubes are highly mathematical puzzles with a vast assortment of existing software, the task of transferring the cubes state into the computer has historically been quite tedious. Here we approach this problem from a computer vision perspective: how might one go about automatically recognizing and extracting the state of a Rubiks cube at any given point in time using a front-facing web camera on commercial hardware, robust to a variety of lighting conditions, scales and motion? We reason that since the cubes face is highly structured and presents a distinct pattern and texture, this problem is an excellent candidate for convolutional neural networks. We present an approach that relies upon two separately trained convolutional neural networks for (1) localization of the cube within live video feeds and (2) subsequent detection of the centers of the cube faces from cropped images. In addition, we present a set of techniques grounded in classical computer vision to extract the Rubiks cube state from the output of the aforementioned convolutional neural networks, featuring SLIC superpixel segmentation, projective transforms and color histogram classification. Finally, we present a technique for gathering large amounts of labeled data for similar problems relying upon an iterative training then assisted labeling procedure and analyze the resultant performance gains. Our code is open source and can be viewed at `https://github.com/jayhack/ConvCube`

1. Introduction

Rubiks cubes have recently emerged as a common task for computer vision researchers to attempt to solve, perhaps due to their neat, structured geometric properties, phyiscal distinctiveness/salience and the practical implications of high-performing solutions. They offer distilled, constrained and easier versions of many hard problems in computer vision, such as localization/detection, geometric reconstruction and pattern recognition.

For one, Rubiks cubes have an internal mathematical structure that has been well-studied and is thoroughly understood by the mathematics community. We know, for instance, an upper bound on the maximum number of turns required to move from any given configuration to another - known as Gods number, [5] it was recently proven using a cluster of computers donated by Google that it takes no more than 20 twists of the cube to get from any given configuration to another. [1] There are simple and deterministic algorithms for traversing the from one configuration to another, many of which are available as well-documented open source software. Their geometric properties make them particularly enticing for computer vision researchers. For one they have a highly distinctive texture, which lends itself well to detection: due to the number of intersections of contrasting colors on the faces of the cube, there is a high probability of finding interesting points to track and distinctive points of high gradient in the x and y directions in image space.

In this paper, we present an approach to extracting a Rubiks cubes configuration that relies heavily on recent advances in convolutional neural networks. Loosely stated, we extract a Rubiks cubes configuration in the following steps:

1. Apply a convolutional neural network to raw images from a webcam feed in order to infer (1) the most likely x,y coordinates of the cube in the image, if it exists, and (2) a scale factor describing how large the cube appears in the image

2. Crop a region from the original, raw image that encloses the point and surrounding region containing the cube inferred by the convolutional neural network mentioned above

3. Apply a second, independently-trained set of three neural networks to infer the most likely locations for the centers of the three faces being presented. This takes advantage of the fact that, since the white and

yellow sides of the cube are always opposite of each other, the camera will see at most one of them, and therefore a single classifier trained to detect either a yellow center piece or a white one will never have to choose between equally qualified candidates. The same is true of blue/green and orange/red.

4. Apply further preprocessing steps to the extracted region, including Gaussian smoothing and Kmeans

5. Extract superpixels from the image using the SLIC superpixel algorithm [1]; gather features on each of these, then classify each in order to locate candidates for rubiks cube pieces

6. Take the centroids of the detected superpixels corresponding to Rubiks cube pieces, orient them around the detected centers described in (3), and run RANSAC in order to find the most likely projective transform describing the relationship between the cube face as seen in the image and the cube face in a flat plain facing the user

7. Apply the most likely projective transform, found in (6), then apply a supervised machine learning algorithm to extract colors from the resultant structured image.

8. Use the results of this algorithm, applied to images in the real world, to gather more data (of potentially lower quality than hand-labeled data) to retrain the numerous machine learning algorithms mentioned above; repeat.

2. Background / Related Work

While there is a general interest in this problem in the computer vision community, it is less of a formal academic pursuit and more of an applications-oriented, project-based engagement from the community. While we have encountered several projects that attempt to perform the same task as we have outlined above, to our knowledge, none of them have either attempted to (1) perform it at the variety of scales that we have gathered data for, or (2) used convolutional neural networks in order to perform this task. We give a brief review of the projects below:

In [4], Kasprzak et al. formulate a computer vision pipeline for extracting a structure representation of a Rubiks cube from images with an eye towards applications in robotics. They describe a process that is capable of extracting a structured representation in real time from single, isolated images. Their approach draws heavily upon colorspace manipulations such as thresholding pixels in YUV space based on hand-picked thresholds in order to locate cube pieces and determine their colors. Notably, they constrain the problem to a single lighting condition and background, which is perhaps appropriate for constrained robotics applications. While they fail to report figures describing their accuracy, they report that their algorithm runs in 90ms on a full-size image on modern computer hardware.

While there is a lack of academic documentation on it, there are a number of available videos on YouTube and otherwise that show individuals demonstrating their own systems. These seem to rely on three particular techniques: edge detection, hough transforms and superpixel detection. The hough line transform is a technique whereby points in image space are transformed into sets in a hough space, where any given point in the hough space corresponds to a line in image space. Lines in the original image that have several salient points on them (such as edge pixels) will then have corresponding peaks in their corresponding hough space. [3] This is used in common practice in a variety of areas and is particularly applicable in this domain as a Rubiks cube is guaranteed to have several detectable lines with known relative orientations. A second commonly used technique involves image segmentation and classification of image segments. One particularly high-performing and applicable image segmentation technique is the SLIC superpixel algorithm, which decomposes an image into a set of roughly rectangular superpixels that are maximally internally consistent in terms of pixel color. [4] This is highly efficient and, along with the hough line transform, is implemented in a number of open source computer vision libraries, including scikit-image and OpenCV.

3. Approach

Here we describe our approach, algorithms, methodology and dataset used for this project.

3.1. Cube Localization via Convolutional Neural Networks

The first, and perhaps most novel step in our algorithm is the application of a convolutional neural network to extract an image region that tightly encapsulates any Rubiks cube present in the image. This is key to our approach as (1) it allows us to ignore noisy and unpredictable background image content in later steps, which are potentially sensitive to noise, and (2) it is a relatively mild up-front investment of CPU time that radically reduces the amount of processing power required to compute subsequent results.

We trained a six-layer convolutional neural network to regress from raw input images to a set of four real numbers approximating the (x,y) coordinates of the center of any cube present, as well as x and y scale factors describing how large the smallest possible bounding box entirely enclosing the cube is. Our final, highest-performing architecture was as follows:

Layer 1 Conv-Relu-Pool 16 3x3 filters; Max pooling; Pool stride of 2, kernel of 2.	**Layer 2** Conv-Relu-Pool 16 3x3 filters; Max pooling; Pool stride of 2, kernel of 2.	**Layer 3** Conv-Relu-Pool 32 3x3 filters; Max pooling; Pool stride of 2, kernel of 2.
Layer 4 Conv-Relu-Pool 32 3x3 filters; Max pooling; Pool stride of 2, kernel of 2.	**Layer 5** Affine-Relu 64 neurons	**Layer 6** Affine 4 neurons

Our raw input data consists of 640x360 RGB images captured from a Macbook Pros web camera in a variety of lighting conditions across the Stanford campus, including indoors, outdoors, dimly-lit rooms, night time and in direct sunlight. We downsampled the images to 100x100 pixels for this step in order to both boost accuracy (we empirically determined that, at least for the architecture described above, downsampled resolution improved accuracy) and significantly cut down on the computational cost We initially trained our net against hand-labelled ground-truth bounding boxes from the gathered images using a euclidean loss function at the final layer. This step is generally very high-performing and works much faster than real time in a variety of lighting conditions. See 3.I for a discussion of our empirical results.

3.2. Finding Cube Pieces via SLIC Superpixel Classification

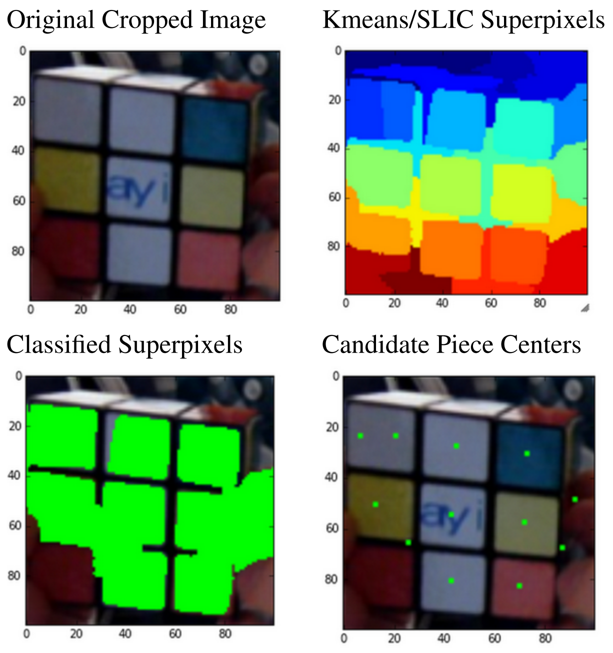
Since the Rubiks Cube (at least the standard one) is guaranteed to have each piece of uniform color, and uniform size and shape, we reasoned that a reasonable way to go about detecting these entities within an image was through (1) segmenting the image into roughly rectangular superpixels using the SLIC superpixel segmentation algorithm, then (2) classifying these superpixels based on internal color histograms and shape descriptors. Concretely, our detection pipeline was as follows:

1. Given an image, run MiniBatch Kmeans on pixel values and assign each pixel to its corresponding centroid to encourage clustering of similar pixels. (This helps with superpixel segmentation)

2. Run SLIC superpixel segmentation on the resulting Kmeans image, producing a set of candidate superpixels for cube pieces

3. For each candidate superpixel, we collect a histogram of HSV pixel values within and calculate up to the third moment of the covered area, appending all into a feature vector

4. Based on a random forest classifier trained on hand-labelled ground-truth data, we then classify all resulting superpixels as cube piece or otherwise.



3.3. Pinpointing Centers of Cube Faces

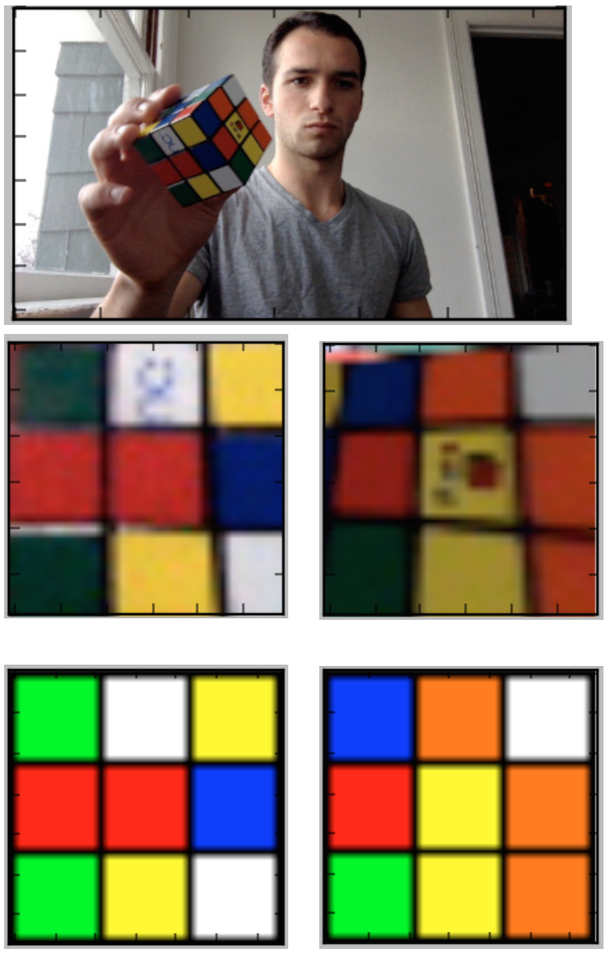
As the images above demonstrate, we achieved reliable performance in identifying locations of candidates for cube pieces; this does not by itself, however, constitute a full reconstruction of the Rubiks cube. In order to understand the cubes structure, one must find point-wise correspondences between locations in the image and pieces of a Rubiks cube. Our solution consisted of the following: employ a convolutional neural network to identify the locations of center pieces of each visible face, then assign the surrounding candidate cube piece centers their cube coordinates based on their relative position to the identified face centers. We trained a six layer network with the exact same architecture as the network described in 3.I, the only change being a final layer that regressed to two coordinates. We trained three of these networks, one each to predict the centers of yellow/white faces, blue/green faces and orange/red faces. This takes advantage of the fact that, since yellow/white are always on opposite sides of the cube, they will never been seen together in the same image and therefore do not require separate entities.

3.4. Bootstrap Learning for Data Acquisition

Since the algorithm outlined above runs in real time and the machine learning portions were clearly benefiting from any additional data that we could offer it, we reasoned that using the output of the various classifiers on real, novel data as additional training data for future iterations would vastly improve performance. To that end, we created a module that allows one to very quickly (1) record themselves with the Rubiks cube in a given location, then (2) immediately after step through the recorded footage and sanitize the output of the classifiers, in order to discard inferences/labels that are way off of the mark and keep our training data set high-quality and precise. End-to-end, recording of a ten-second-long video (roughly our average video length) and hand-labelling it takes approximately 15 minutes and produces between 150 and 300 labelled data samples. (The majority of the time in hand-labelling images comes from painstakingly drawing bounding boxes on each frame.) The bootstrap learning process, in contrast, merely asks one to confirm or reject labels that the algorithms themselves have produced, which is near instantaneous for each frame - this cuts down the time to label a ten second video to a matter of two to three minutes.

3.5. Extracting Cube Face Information

Given our predictions for the centers of the cube faces (and specifically, the corners of the center cube), we were able to calculate a projective transform to orient the cube and extract the color information from the faces. We were then able to classify the colors of each square by using a manually-tuned SVM.



4. Experiments/Results

4.1. Cube Localization via Convolutional Neural Networks

To train the network described in 2.I for cube localization, we gathered and hand-labelled 1,350 frames from 15 different videos with ground-truth bounding boxes tightly bounding the cube. We intentionally gathered videos in a wide variety of lighting conditions and background environments, including several indoor and outdoor locations around campus both in the afternoon and in the evening. In addition, we intentionally gathered videos that frequently show the cube in motion, partially occluded by the hand, and at multiple scales, up to roughly five feet from the camera lense. These videos were all shot with the front-facing camera on a Macbook Pro at 640x360 resolution, then down-sampled during training to 100x100 to both increase accuracy and improve on the amount of time required. Furthermore, in order to increase the amount of training data available, we flipped every frame around the y axis and adjusted contrast with scalar constants randomly sampled from the interval (0.6, 1). Our best training and validation loss on this data set are summarized below. (Note that coordinates and predictions were on a scale of 0 to 1, so a validation loss of 0.5 indicates that we were routinely halfway across the image from the true label.

	Train Accuracy	Validation Accuracy
Best 4-Layer Net	0.034	0.038
Best 5-Layer Net	0.035	0.032
Best 6-Layer Net	0.028	0.035

These results were attained with a data set of over 4,000 frames from 27 videos, over half of which had been attributed labels in our bootstrap labelling process.

One of our most significant discoveries in the development of this algorithm was that the forward pass for even the 6-layer net, implemented in Python with Cython extensions (using code from CS 231N), took merely 7 milliseconds to run. This is far faster than is necessary for real time, especially considering it was the most time-consuming portion of our entire classification pipeline.

4.2. Finding Cube Pieces via SLIC Superpixel Classification

In this classification pipeline, the most costly step by far was the assignment of pixels to their corresponding kmeans centroids, which took an average of 5ms to perform using MiniBatch Kmeans implemented by scikit-learn in python. We trained a Random Forest classifier with 50 base estimators on roughly 2,500 hand-labelled examples of superpixels, roughly 20% of which were positive examples. We achieved an average of 92% classification accuracy at a threshold of 20 trees voting positively during k-fold cross validation with five folds.

5. Conclusion

Through the application of an assortment of convolutional neural networks and other classical computer vision methods, we have created an application that can track a Rubiks cube in real time across a video at a variety of scales and lighting conditions. While our current implementation does not support structured extraction of face information in real time, we fully intend to continue developing this project going forward and believe it is only a matter of implementation, as our algorithms are performing objectively well offline, as can be seen above. To our knowledge, we are the first to apply convolutional neural networks to the problem of tracking Rubiks cubes in real time and extracting structured representations from them, and have round multiple applications (localization, finding the centers of faces) in which they perform objectively well. Noting that the number of potential applications for convolutional neural nets is huge, we believe that other researchers investigating this domain would benefit from experimenting with shifting as many task as possible to convolutional neural networks as possible, such as the segmentation and superpixel classification phase (CNNs work extremely well for image segmentation, as Lecun et al. have shown in [2]) and even perhaps recognizing the roll, pitch and yaw of the cube directly. We also believe that there is interesting work to be done in constructing a hough transform specifically for Rubiks cubes - that is, one could map the centers of cube pieces to a hough space representing all possible (x, y, z, roll, pitch, yaw, scale) cubes within an image, then extract peaks from this space. This is a more classical approach and would seemingly require orders of magnitude less data in order to become operational.

We would like to thank the CS231N Winter 2015 staff for the CNN code that they developed for homework assignments, which we have also used in our project. We have open sourced all of our code (still currently under active development) on github and encourage the computer vision community to contribute ideas and code. You can access the code here: `https://github.com/jayhack/convcube`. Finally, we are convinced that the core models we have constructed have enormous potential and are committed to continue amassing large amounts of labelled data through the bootstrapping procedure we have described above in order to further improve its performance.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Ssstrunk. Slic superpixels. EPFL Technical Report no. 149300, June 2010. Available at `http://ivrg.epfl.ch/research/superpixels`.

[2] J. Alvarez, T. Gevers, T. LeCun, and A. Lopez. Road scene segmentation from a single image. `http://yann.lecun.com/exdb/publis/pdf/alvarez-eccv-12.pdf`.

[3] J. Illingworth and J. Kittler. A survey of the hough transform. Science Direct Volume 44, Issue 1, October 1988, Pages 87116. Available at `http://www.sciencedirect.com/science/article/pii/S0734189X88800331#`.

[4] W. Kasprzak, W. Szynkiewicz, and L. Czajka. Rubiks cube reconstruction from single view for service robots. `http://www.ia.pw.edu.pl/~wkasprza/PAP/ICCVG06.pdf`.

[5] T. Rokicki and H. Kociemba. God's number is 20. `http://www.cube20.org`.