DIVERSIFIED VISION-BASED PUZZLE SOLVER

Andrew Corum

Indiana University Bloomington

ABSTRACT

A touchstone of intelligence, puzzle solving and game playing has been a desired skill to pass on to computers for many years. In parallel, there has also been significant effort to program computers to perform object detection and extract information from images. This project seeks to solve a problem at the intersection of these two areas, and solve a variety of puzzles given only an image. In this developed method, an image classifier is first used to detect puzzle type. Then various computer vision techniques and OCR to extract information from the pictured puzzle. Finally, classic artificial intelligence search algorithms can determine the solution to the puzzle, which is outputted to the user. While this method may be made more robust to environmental factors such as lighting or occlusion, the project result still shows good results.

***Index Terms*—** Machine Learning, Computer Vision, Neural Network, Artificial Intelligence

1. INTRODUCTION

The use of games and puzzles has become a quintessential component in the search for general artificial intelligence. Puzzle solving has been a classic measure of intelligence. Top research groups continuously work to improve computers' abilities to solve classic puzzles such as Chess and Go [1, 2]. Thanks to the versatility of abstraction, the theory behind solving puzzles and games can be more than just basic measures of intelligence or pedagogical tools [3], but even used to tackle real world problems as far reaching as cryptography, biology, and economics.

Computer vision is a seemingly distinct area of computer science that has received significant attention in recent years. Some classic computer vision problems are object detection/recognition, image classification, and segmentation. Computer vision systems often employ multiple techniques to solve a range of complex problems, such as autonomous driving or facial recognition.

The problem that this project specifically tries to solve falls directly in the intersection of computer vision (CV) and game solving. When given an image of an arbitrary logic puzzle, we would like to be able to solve the puzzle. However, this goal is too general for current AI and ML techniques, so it can be simplified to solving a only a selected handful of puzzles. This project seeks to be able to solve Sudoku, Rubik's, and Akari puzzles (see Figure 1).

(a) Sudoku

(b) Akari

(c) Rubik's Cube

Fig. 1. Examples of puzzle input images.

2. APPROACH

The primary goal of this project is to construct a program that can solve a puzzle when given as a picture/image. Additionally, rather than only solving one specific type of puzzle (which is often done in other projects [4, 5]), this project should be capable of solving a diversified range of logic puzzles. Since a general artificial intelligence puzzle solver may be impractical given the current state of AI/ML, this diverse puzzle solver will instead be designed to handle a set of prede-

termined puzzle types (ie Sudoku, Rubik's cube, Akari, etc).

The diverse vision-based puzzle solver follows a pipeline to solve a given image of a puzzle, as shown in Figure 2. A puzzle will be pictured and passed into an image classifier. This classifier will need to be trained on a dataset containing images of labeled puzzles. Features of the puzzle, such as lines or digits, will need to be extracted depending on the puzzle type. Once the necessary information has been extracted from the input image, then the puzzle can be solved using various AI techniques, and then the solution visualized/presented.

2.1. Puzzle Classifier

Modern image classifiers often apply convolutional neural networks (CNNs); however, this was not a feasible option for this particular project. One of the challenges of arbitrary puzzle classification is that there are not datasets of arbitrary labeled puzzle images available. As CNNs require huge amounts of data, they are impractical here. Instead, we can use some computer vision feature extraction techniques and classic ML algorithms (such as KNN or Naive Bayes). Figure 3 shows the process used in this project to classify images.

To extract features from each input image, we can apply the Scale-Invariant Feature Transform (SIFT) [6]. This transform deterministically detects keypoints and their descriptors within an image. Then these descriptors can be used as a kind of "bag of words" to create a histogram that describes the input image. This is a technique that is commonly used in both computer vision and natural language processing. The SIFT feature extraction and bag of words preprocessing needs to be applied to both the input image and the training data that is used to construct the classifier. When training the classifier, KMeans clustering can be used to find which clusters of descriptors (ie "words") best define the puzzles we are studying.

Again, there is no authoritative data source on images of various puzzles, so this project uses a hand-created dataset of 44 images. The implemented image classifier uses KNN to learn features from these images, then predict the puzzle type of an incoming image. This project also implements Naive Bayes to classify puzzle type (but with no boost to prediction accuracy).

2.2. Puzzle Features

After the puzzle has been classified, we need to extract data and features from the image. This involves a variety of ML and CV techniques once again. Additionally, each puzzle type will need its own specialized feature extraction methods.

2.2.1. Optical Character Recognition

A necessary piece of information that is part of both Sudoku and Akari puzzles comes from written/printed digits on the page. In order to identify which digits are printed on a puzzle, we need a classifier that can perform optical character recognition (OCR). The MNIST dataset [7], containing 70,000 images of 28x28 handwritten digits, can be used to train the OCR model. Since we have access to a larger dataset of images, this is a good place to apply a CNN. This project uses TensorFlow to construct and train a multi-layer CNN. The model is trained on 60,000 MNIST images then tested on the remaining 10,000 (see section 3 for more info on the OCR CNN evaluation).

2.2.2. Sudoku Features

The Sudoku feature extraction process is pictured in Figure 4. When given a Sudoku image (like the one shown in Figure 1), the first step is to attempt to identify the puzzle's exact location. This can be done by detecting the Sudoku's grid. First we perform Canny edge detection [8] and then the Hough transform to identify lines on the image [9] (results of this step in the process are shown in Figure 5). Next we can identify the corners of the grid, and perform a warping/homography to isolate the Sudoku.

Since the Sudoku is known to be a 9x9 grid, we can iterate over each of the 81 squares in the isolated Sudoku image, determining which is empty or non-empty by using a simple grayscale threshold. For the cells that contain digits, we then apply our OCR CNN classifier (introduced in section 2.2.1) to classify each digit. At the end of this process, we now know which digit is at each location on the board, and can even visualize the detected Sudoku data, as shown in Figure 6.

2.2.3. Akari Features

The Akari feature extractor reuses much of the processes implemented in the Sudoku feature extractor. Canny edge detection, Hough transform, and homography are still used to isolate the Akari grid. We can assume a 10x10 Akari board, then search each cell for digits. The only difference is that the Akari feature extractor must also identify the difference between the black cells and white cells, which can be done using a naive threshold on the average grayscale values.

2.2.4. Rubik's Features

Extracting data from an image of the Rubik's cube is a bit tricky. This is because it is not realistically possible to capture the whole cube in just one image. This project circumvents the issue by requiring multiple images of the cube to be provided once a Rubik's cube is detected. The additional images should be presented so that each face is pointed directly towards the camera.

Figure 7 shows the process for extracting features from an image of a single face of the Rubik's cube. Since we only consider certain colors when looking at a Rubik's cube, simple thresholding is done over the HSV (hue, saturation, value) pixels in the image. Then contours of the threshold image can

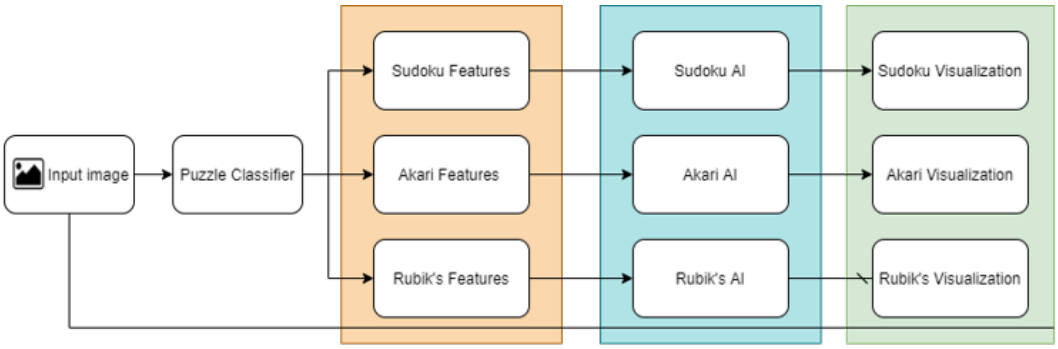

Fig. 2. Pipeline for diverse vision-based puzzle solver.

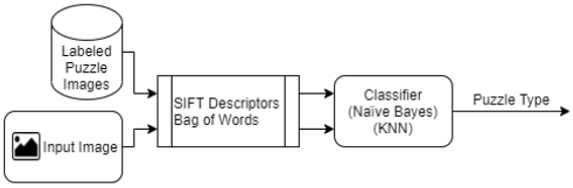

Fig. 3. Puzzle-type classification process.

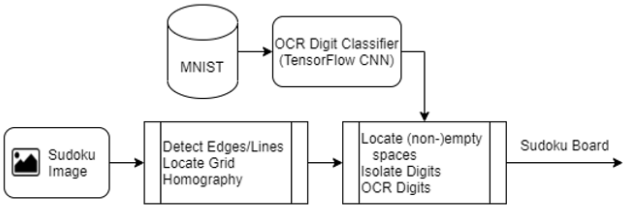

Fig. 4. Sudoku feature extraction process.

be identified. From these contours, only the ones that are most "square-like" should be considered. "Square-like" contours are identified by determining the bounding box of the conour and determining if the area within the contour closely approximates the area within the bounding box. Finally, after the Rubik's cube squares are identified, clustering is performed to identify which color label best fits.

2.3. Artificial Intelligence

In order to actually solve the puzzle, artificial intelligence (AI) techniques can be used. To solve the Sudoku puzzle, this project implements a simple brute-force search algorithm. The Sudoku search space is constrained enough that this can find a solution in a very reasonable amount of time (1 or 2 seconds). Initially, a similar algorithm was used to attempt to solve the Akari problem; however, there are fewer initial constraints on locations where the "lights" should be placed on the Akari board. To get around this, the puzzle solver uses a python library for constraint-based programming, which has more a finely-tuned search algorithm to find solutions given properly defined constraints. To solve the Rubik's cube, this project uses a library called "Kociemba" [10]. This uses a two-phase search algorithm to reasonably quickly find a short solution to the cube.

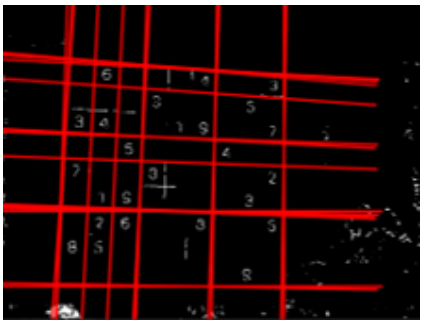

Fig. 5. Sudoku grid lines detected by the Hough transform.

Fig. 6. Visualized Sudoku features.

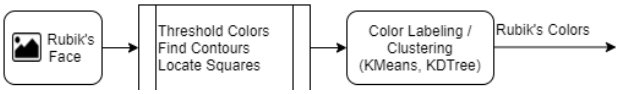

Fig. 7. Rubik's feature extraction pipeline.

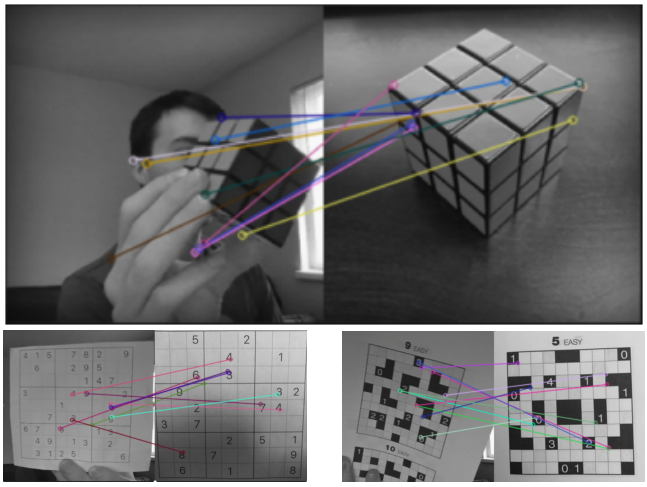

Fig. 8. Real-time puzzle-type classification results, showing matched keypoints/features.

3. EXPERIMENTATION

Another problem of having limited data is that experimentation becomes quite difficult. Various ML techniques are used throughout both the classification (section 2.1) and feature extraction (section 2.2) steps of this project. When using ML techniques, it is important to find ways to evaluate the models performance. To experiment and evaluate the puzzle-type classifier's performance, standard cross-validation or test-train split will not suffice. There is simply not enough data to cause any error rate in the classification. Instead, we can use a live-feed webcam and view repeated classifications over multiple frames of video. Figure 8 demonstrates a few frames of the classifier correctly identifying which puzzle-type is being presented to the webcam.

Experimenting with the OCR CNN, thanks to the large MNIST dataset, was a much easier task. The data could be split into 60,000 training images and 10,000 test images. Then, over multiple iterations, the CNN TensorFlow architecture was modified to find one with optimum performance. Figure 10 shows the training curves of two different CNNs that were tested during this experimentation process. The first, called TinyCNN, is a very small network without many features (such as dropout or batch normalization). As is evident in Figure 10, TinyCNN does not quite have the same performance as the second OCR CNN. The second OCR CNN, which is used in the final version of this project, is the result multiple iterations of development, and has high

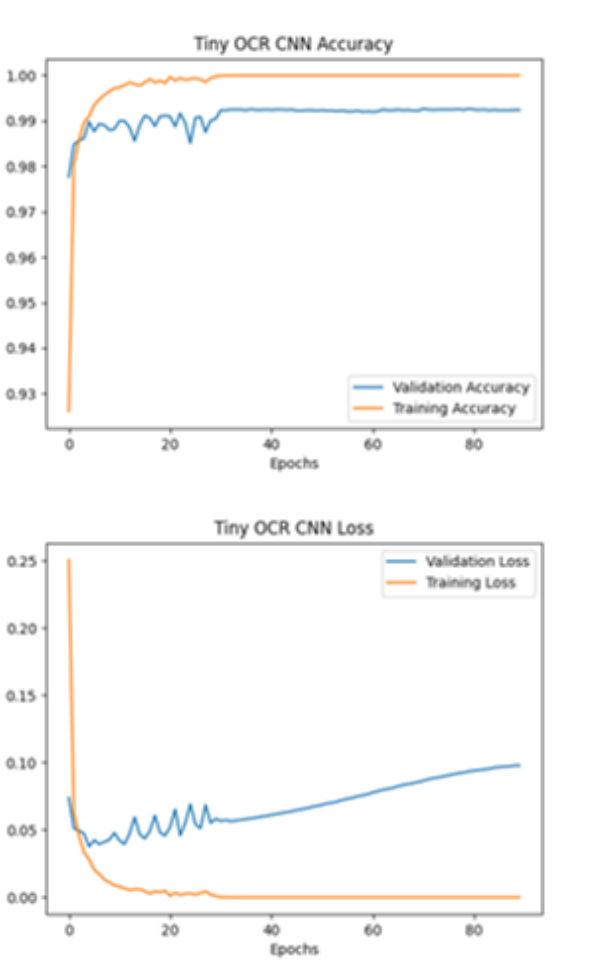

Fig. 9. Learning curves of TinyCNN OCR classifier.

performance on the training MNIST data.

4. RESULTS

This diverse puzzle-solving program performs fairly well... when the image conditions are reasonable. Figures 11 and 12 show the visualized output. In each image, the solution is properly computed and displayed over top of the cropped, original puzzle.

The performance of this puzzle solver pipeline is greatly affected by lighting. The Rubik's cube feature extractor is not able to correctly identify colors unless the lighting in the scene is nearly perfect. Unsupervised clustering of colors like yellow and orange becomes very difficult when lighting varies from one image to the next.

Additionally, the problem with using a pipeline approach to this problem is that, if one portion of the pipeline fails, then the entire puzzle solution will be incorrect. A lot of work went into tuning and optimizing the feature extraction and OCR methods. If a single digit is missclassified, or the

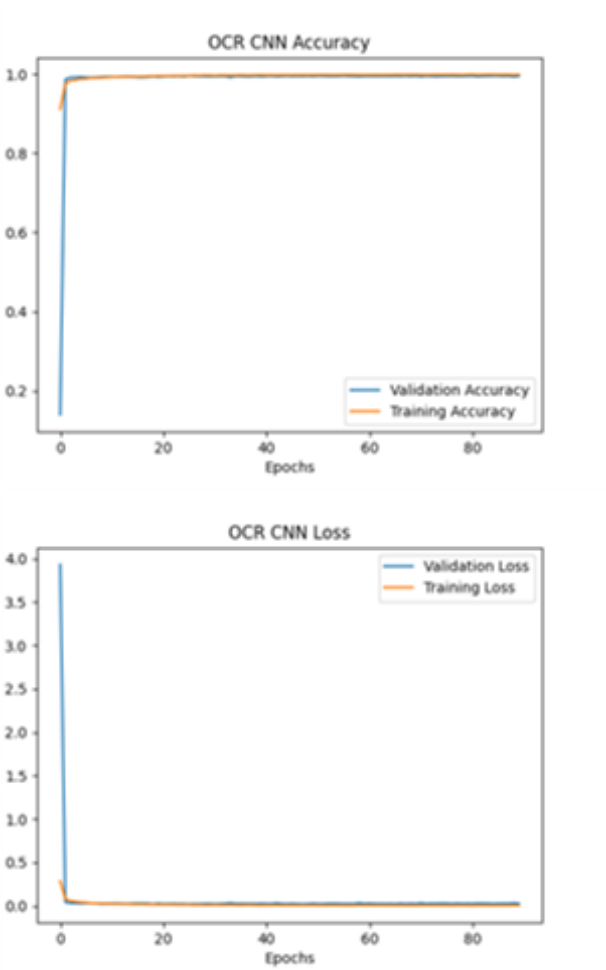

Fig. 10. Learning curves of improved OCR CNN classifier.

Fig. 11. Solved Sudoku puzzle.

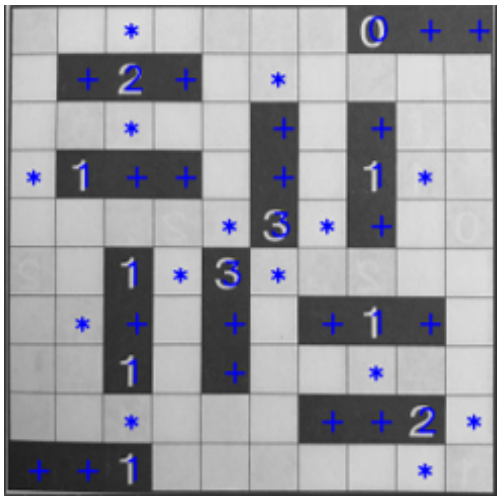

Fig. 12. Solved Akari puzzle.

puzzle homography is not properly computed, then the rest of the solution will fail.

5. CONCLUSION

The results of the diverse vision-based puzzle solver are very satisfying. When given an image, this program is *often* able to properly identify which puzzle is pictured, then perform the necessary feature extraction and AI steps to compute the solution. However, there are some drawbacks. The pipeline method leaves a lot of opportunity for error. In ML and CV projects, it is always best if you can model the *entire* problem at once, rather than strictly one step at a time. For example, if the feature extractor could not detect a valid Sudoku, then this information could be fed back to the puzzle-type classifier to aid in its decision.

Additionally, this solution is not entirely robust to changes in the scene of the input image. If there are many other lines, lights, and colors in the background of the image, this program will have great difficulty properly grabbing the necessary data to evaluate the puzzle. Perhaps more advanced computer vision techniques could help mitigate some of these issues.

Altogether, this project showcases how various ML and CV techniques can be applied to allow a computer to solve a seemingly daunting task. Using data cleansing, feature extraction, classification, object recognition, and convolution, this project constructs a program that can solve a small group of puzzles given only an image.

6. CODE REFERENCES

Various non-peer-reviewed resources were used when developing this project. These references can be found in the project `README.md` file.

7. REFERENCES

[1] Feng-hsiung Hsu, "Ibm's deep blue chess grandmaster chips," *IEEE Micro*, vol. 19, no. 2, pp. 70–81, 1999.

[2] Fei-Yue Wang, Jun Jason Zhang, Xinhu Zheng, Xiao Wang, Yong Yuan, Xiaoxiao Dai, Jie Zhang, and Liuqing Yang, "Where does alphago go: From church-turing thesis to alphago thesis and beyond," *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 2, pp. 113–120, 2016.

[3] Z. Markov, I. Russell, T. Neller, and N. Zlatareva, "Pedagogical possibilities for the n-puzzle problem," in *Proceedings. Frontiers in Education. 36th Annual Conference*, 2006, pp. 1–6.

[4] S. Gur and O. Ben-Shahar, "From square pieces to brick walls: The next challenge in solving jigsaw puzzles," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 4049–4057.

[5] Baptiste Wicht and Jean Hennebert, "Camera-based sudoku recognition with deep belief network," *6th International Conference on Soft Computing and Pattern Recognition, SoCPaR 2014*, pp. 83–88, 01 2015.

[6] David G Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*. Ieee, 1999, vol. 2, pp. 1150–1157.

[7] Li Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[8] John Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, , no. 6, pp. 679–698, 1986.

[9] John Illingworth and Josef Kittler, "A survey of the hough transform," *Computer vision, graphics, and image processing*, vol. 44, no. 1, pp. 87–116, 1988.

[10] "Kociemba," 2016, https://github.com/muodov/kociemba.