Progress Report: Diversified Vision-Based Puzzle Solver

Andrew Corum

April 5, 2021

1 Timeline

Goal	**Due**	**Done?**
Create very small dataset for a few puzzle types Currently have total of 33 images for Rubik's, Akari, SlitherLink. Also have a script, `video_to_data.py`, that can convert my webcam video feed into labeled data.	N/A	✓
Construct basic classifier(s) for given puzzle types Current classifier uses SIFT keypoints/descriptors and Bag of Visual Words technique to classify Rubik's, Akari, and SlitherLink puzzles. Both KNN and Naive Bayes models have been tested. See `classifier.py`.	N/A	✓
Classify puzzles given live video feed `main.py` uses the webcam feed to classify puzzles in real time, as well as display some matching keypoint descriptors.	N/A	✓
Progress report	April 5	✓
Data/feature extraction (for at least one puzzle type)	May 12	
Classic AI solution (for at least one puzzle type)	May 12	
Advanced ML solution (DNN, CNN) (for at least one puzzle type)	May 19	
Visualization of puzzle solutions (for at least one puzzle type)	May 19	
Additional puzzle(s) data/feature extraction	May 26	
Additional puzzle(s) solution and visualization	May 3	
(Optional) Data augmentation to improve puzzle classification	May 3	
Final submission	May 4	
Project presentation	May 4, 6	

2 Progress

The timeline above gives an overview of the project's current progress. I have constructed a small dataset for three different puzzle types. It is clearly not large enough for training any kind of neural network, but it seems fairly robust when used by the Bag of Visual Words technique [5]. I create *visual words* by computing SIFT keypoints and descriptors [2a] and then applying KMeans [2c] to find clusters of keypoints in the training data. Then each test image can be represented as a histogram of the closest clusters/visual words, which is analogous to the standard Bag of Words strategy. Finally, I apply a simple classifier (either KNN or Naive Bayes) to the *visual words* histogram to classify any given test image. The code for this can be found in `classifier.py`.

Sample classifier.py **output:**

```
$ python3 classifier.py
Grabbing images
SIFT
Cluster
Hist
Hist2
KNN
Accuracy: 0.9090909090909091
Confusion Matrix:
[[4 0 0]
 [0 4 0]
 [0 1 2]]
```

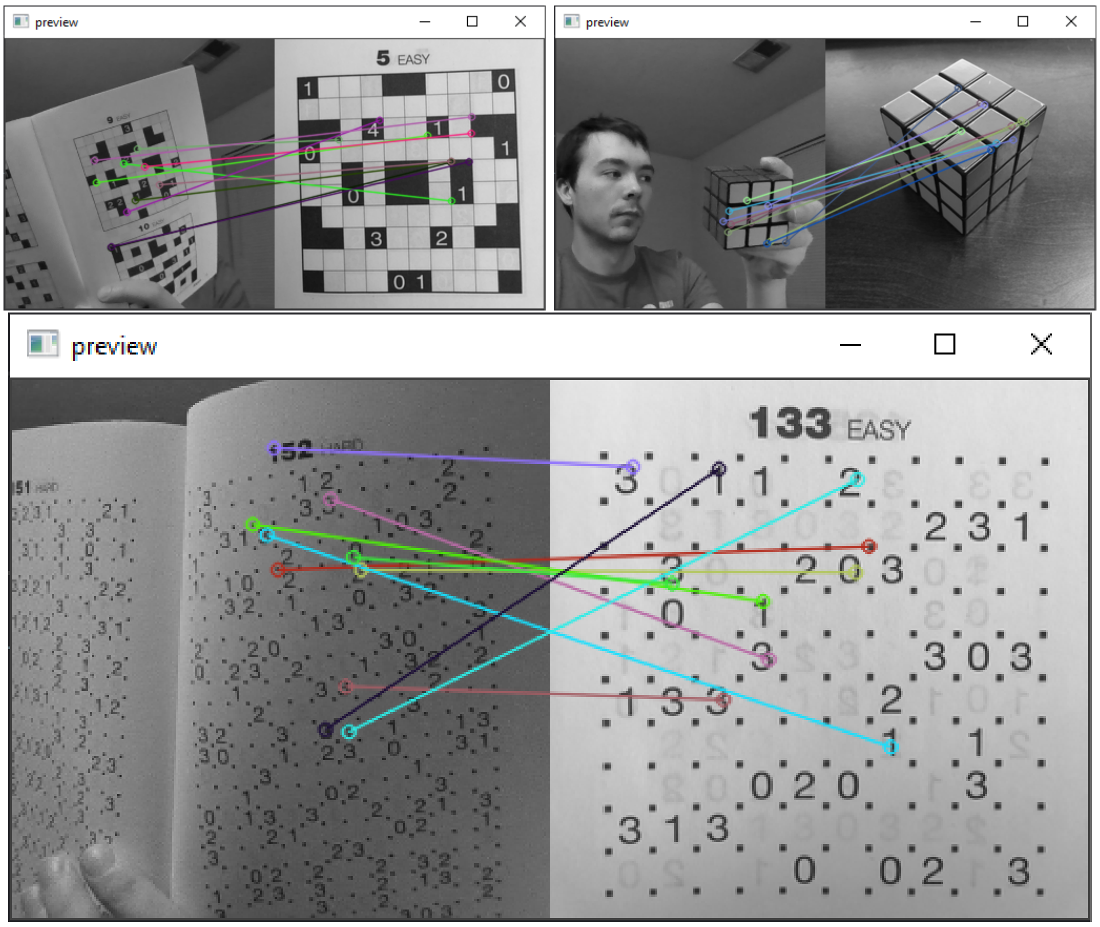

Figure 1: Live display of puzzle classification in `main.py`, including some matching keypoints.

The `video_to_data.py` program is used to convert webcam feed to labeled data. It is fairly simple and does not necessarily produce diverse data. This is because the background of the webcam feed is whatever environment I am currently in (ie my apartment). But the results are good enough for the simple classifiers.

The `main.py` script uses live feed from the webcam to show the puzzle classification in real time. It builds a model using all training data found in `./project/data`, then uses the model to predict which puzzle is being shown in each given frame. The program then displays the current webcam frame as well as the predicted puzzle type with its matching keypoint descriptors. Figure 1 shows examples of this output.

3 Q and A (from proposal)

Q:

"Hi Andrew, this project is interesting, but it also seems ambitious, as you eluded too. Basically, I have the same concerns as you about the data. Is there a publicly available dataset that you can leverage? Is there an online version of the puzzle/game which may have an API that you can use?"

A:

Based on my current progress, I believe the scope of this project is reasonable. This is, in part, because I have set multiple milestones that will yield interesting results. I intentionally designed my timeline (see above) to ensure that I still have presentable results even if my entire goal is not completed.

The effect of limited data varies between each step of this project. For example, I have successfully created a simple 3-class puzzle classifier with my hand-made data. However, creating a more complicated CNN classifier is likely out of reach for this project. It would require a lot more data, and probably advanced data augmentation / synthetic data techniques.

I have found a couple sources of public data that may be useful for the puzzle solving step: `https://www.kaggle.com/rohanrao/sudoku` and `https://github.com/adrianliaw/pycuber`. There are some online APIs that I could use, but it should not be necessary. These puzzles are conceptually simple, and I am comfortable coding a basic API on my own.

Q:

"It probably will be best if you select a single type of puzzle (e.g. Sudoku) and start from there to narrow the focus."

A:

This is essentially my plan. I do not want to throw out the puzzle-type classifier altogether (I am interested in learning more about image/object classification). However, I plan to focus on only one puzzle at a time. As a bare minimum, I plant to have at least one puzzle able to move through my entire project pipeline (classification, feature extraction, and solving).

Q:

"Also, what exactly will the output be? Is it just the solved problem (e.g. Sudoku board), or are you trying to incorporate incremental solutions, as is done with Reinforcement Learning with Go? I think the first option should be where you start."

A:

The output should be the entire solution to the input puzzle. For example, for the Sudoku puzzle, the solution should be the correctly filled in Sudoku board.
But you bring up a good point. For the more complicated puzzles/methods, a sequential approach may be necessary. The Rubik's cube puzzle, in particular, would be extremely difficult to solve in one shot. More likely, I would need to create a model that would determine the best next step. Then the model could be applied repeatedly until the puzzle is solved. This is a more complicated process, and I will only attempt this after I have already completed the more simple tasks (such as solving the Sudoku puzzle).

Q:

"Basically, I'm suggesting that you simplify the problem as much as possible, and incrementally add difficult, as time permits. I expect you to have all of this clarified by Milestone 2."

A:

I agree with you here. I plan on taking the incremental approach, so that I will, at the very least, have a simple puzzle solver to show next month. The additional, more technical tasks will just be *icing on the cake*.

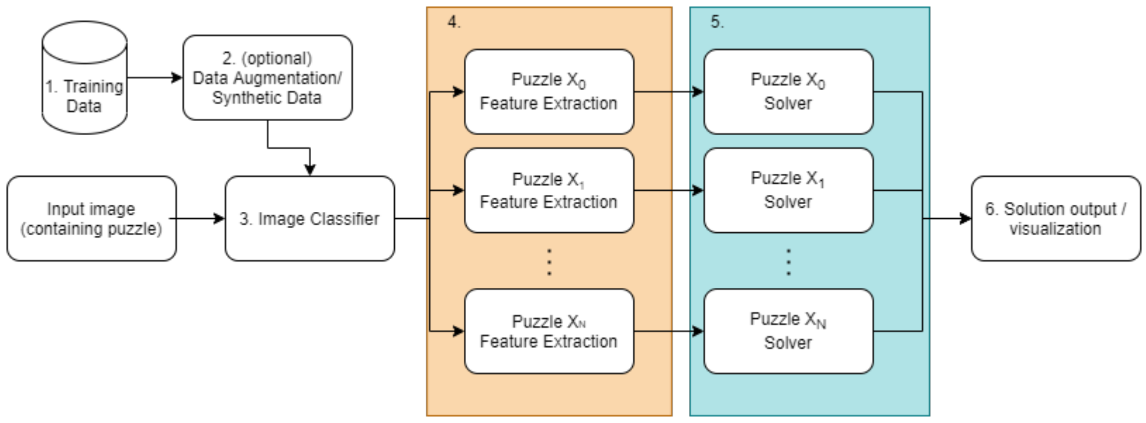
1. Training Data
2. (optional) Data Augmentation/ Synthetic Data
Input image (containing puzzle)
3. Image Classifier
4.
Puzzle X0 Feature Extraction
Puzzle X1 Feature Extraction
Puzzle XN Feature Extraction
5.
Puzzle X0 Solver
Puzzle X1 Solver
Puzzle XN Solver
6. Solution output / visualization

Figure 2: General pipeline for diverse vision-based puzzle solver.
See timeline above for intermediate goals/milestones.

4 References

1. https://imagemagick.org/
2. https://docs.opencv.org/
   (a) https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html
   (b) https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html

   (c) https://docs.opencv.org/master/d1/d5c/tutorial_py_kmeans_opencv.html

3. Webcam using OpenCV:
   https://stackoverflow.com/questions/604749/how-do-i-access-my-webcam-in-python

4. OpenCV crop:
   https://stackoverflow.com/questions/61927877/how-to-crop-opencv-image-from-center

5. Bag of Visual Words:
   (a) https://en.wikipedia.org/wiki/Bag-of-words_model_in_computer_vision
   (b) https://medium.com/@aybukeyalcinerr/bag-of-visual-words-bovw-db9500331b2f

6. Searching for array in a 2d array:
   (a) https://www.w3resource.com/python-exercises/numpy/python-numpy-exercise-171.php

7. https://scikit-learn.org/stable/modules/classes.htm