Puzzle Boon

Jigsaw Puzzle Assist

CSI4900 Honours Project

Tiffany Maynard

300047456

Winter 2021

Supervisor: Dr WonSook Lee

Introduction **3**

Methodology **4**
Initial Exploration 4

Solutions **9**
Creating a dataset of puzzle piece images 9
Train model to locate puzzle pieces in an image 14
Locate corners and orient pieces 14
Various matching strategies 15
Communicate results of matching 17

Results **18**

Conclusion **19**

Future Work **20**

References and Tools Used **20**

Appendix - Code File Index **21**

Introduction

For my honours project, I chose to make a tool to assist with solving jigsaw puzzles.

I got interested in jigsaw puzzles as a means to distract myself from the pandemic and relax.

With more difficult puzzles, sometimes I felt like I had puzzle block, kind of like writer's block, and I was stuck because I couldn't find any pieces to fit within the part I had finished so far. I realized it would be great if there was a tool that could help me find the next few pieces to break the stalemate.
Some puzzles are tough because they have so many pieces and others because they have indistinct colours or non-traditional piece shapes. I have one puzzle that has 18,000 pieces and that will be almost 2 metres by 3 metres in size when complete. Since I will need to dedicate the entire floor of one room to completing it because I don't have any tables quite so large, I have not been brave enough to attempt it yet. It would be a boon to have a helpful tool.

An 18,000 piece puzzle - 2m x 3m in size

A gradient puzzle

I also have other tough puzzles like a silver crypt puzzle where all pieces are an identical silver colour and have unusual circular shapes, and a gradient puzzle where the colour varies only slightly from top to bottom.

Methodology

Puzzle Boon will suggest possible matches based on three photos – one of the full puzzle image from the puzzle box, one of the puzzle in progress, and one of the remaining puzzle pieces which cannot be placed due to acute puzzle block.

Full puzzle image

Puzzle in progress

Remaining puzzle pieces

With this information, the tool will analyse the images and provide suggested matches showing which remaining piece will fit in which part of the puzzle of progress.

Puzzle Boon needed a way to recognize which items in a photo were puzzle pieces, a way to determine likely matches between the blockers and the puzzle in progress and a way to clearly communicate the possible matches to the user.

Initial Exploration

To start, to find puzzle pieces in a photo, I first tried traditional computer vision tools and algorithms using OpenCV, an open source computer vision library that includes a huge number of functions to analyse and manipulate images and video. The traditional approaches I tried required significant manual intervention and they weren't always effective at isolating the puzzle pieces correctly, depending on the lighting and background of the image.

As a result, I decided I would try to use deep learning to train a model to detect puzzle pieces. Deep learning has an advantage over traditional tools because it can be trained with task-specific data by even a naive user like me, while the algorithms used in traditional image processing are not something I am able to customize to be task-specific without significant domain knowledge.

For example, here you can see my attempts at using Sobel and Canny image filters, where they found edges both inside and outside of each puzzle piece. Results were also indistinct with GrabCut when starting from a bounding box around the piece. Similarly, the Hough transform gave false edges and the goodFeaturesToTrack corner detector found way too many corners.

Sobel

Canny

GrabCut with BoundingBox

goodFeaturesToTrack corners

Hough transform

I also tried to do “brute force” feature matching in OpenCV using the BFMatcher class and the ORB keypoint detector and descriptor extractor but it was unsuccessful as the located matches were incorrect.

Unsuccessful result from brute force feature matching

These disappointing results from traditional computer vision led to my decision to use deep learning to find puzzle pieces in an image.

I watched online lectures and read about the different types of visual perception tasks that deep learning can handle including Image Classification, Semantic Segmentation, Object Detection, and Instance Segmentation

Source: Stanford CS231n - Lecture 12 Detection and Segmentation http://cs231n.stanford.edu/slides/2020/lecture_12.pdf

and realized I would need to do instance segmentation, since my images would contain many instances of the same object, in this case puzzle pieces, and I needed to know the distinct location of each piece.

For instance segmentation, Mask R-CNN was chosen. Mask R-CNN (Region Based Convolutional Neural Network) is one available deep neural network architecture that aims to solve instance segmentation problems, so I decided to try it.

Source: Stanford CS231n - Lecture 12 Detection and Segmentation http://cs231n.stanford.edu/slides/2020/lecture_12.pdf

Source: Stanford CS231n - Lecture 12 Detection and Segmentation http://cs231n.stanford.edu/slides/2020/lecture_12.pdf

Models based on Mask R-CNN have already been trained to recognize many things like people, cars, and dogs. Unfortunately Mask R-CNN was not already trained to recognize puzzle pieces, and does not have the facility to recognize them built-in, so I needed to develop my own dataset of puzzle pieces and then train a model to recognize them using this dataset.

Training a model from scratch can take a very long time and require a really large amount of data, so luckily, I was able to leverage an existing trained model by using transfer learning.

Source: Doğru, Anıl & Bouarfa, Soufiane & Arizar, Ridwan & Aydogan, Reyhan. (2020). Using Convolutional Neural Networks to Automate Aircraft Maintenance Visual Inspection. 10.20944/preprints202011.0527.v1

Transfer learning uses a model trained on one task and re-purposes some of the findings for a new task. I began with pre-trained weights that were already good at determining patterns and shapes in images of common objects and used these as the starting point for my puzzle piece training.

Solutions

My solution needed several components, including creating a dataset of puzzle piece images, a trained machine learning model, the means to locate puzzle piece corners and re-orient pieces, some matching strategies, and a way to communicate the match results.

Creating a dataset of puzzle piece images

To develop my dataset, I took lots of pictures of puzzle pieces from many different puzzles on many different coloured backgrounds.

LabelStudio interface view showing many different backgrounds

I manually labelled my images with bounding boxes around each puzzle piece using a free tool called Label Studio. The LabelStudio interface made it easy to keep track of which images had been annotated with bounding boxes and allowed for easy corrections and adjustments. Here are some sample images with the bounding boxes drawn for each piece.

Puzzle pieces with bounding boxes in LabelStudio

Once I had my set of images and their bounding boxes, I exported them to PascalVOC format which included a json file with the bounding box information for each image, and I isolated the

pieces from the background using the background colour within each bounding box as a filter. To find the range of Hue, Saturation and Value (HSV) numbers to filter for each background colour, I loaded an image of each background colour into HSV visualizer and adjusted the HSV ranges until the pieces were shown in relief. I noted these HSV ranges for each background colour in order to use them in the OpenCV inRange function to mask the background.

Image before adjusting HSV sliders

Image after isolating pieces from background with HSV sliders

I used the OpenCV GrabCut algorithm to isolate the foreground and background and create a refined mask of each puzzle piece. A mask is a black and white image of the same dimensions as the original image. Each of the pixels in the mask is either black (0) or white (1). The mask can be used as a filter to keep only the relevant part of an image or to make the contrast more obvious for finding edges and contours.

Original image within bounding box

Mask = filter

Image combined with mask to remove background

To use GrabCut, you designate parts of the image that are definite or probable foreground and background. This can be done using a mask or a bounding box, with everything outside the bounding box being taken as “definite background”. A graph of connected components is constructed with this information and then after the specified number of iterations to refine the graph connections, a “cut” is done to separate the foreground from the background components.

Source: https://www.docs.opencv.org/master/d8/d83/tutorial_py_grabcut.html

I had success with GrabCut when I used the rough masks created by filtering out the background based on the HSV ranges found above as the starting point for the algorithms iterations. This is in contrast to the lack of success with a bounding box starting point described above in the Initial Exploration section of this report.

These GrabCut-refined masks were used to get the precise location of the whole puzzle piece and its outline and to export this information to the COCO format. I used a tool called pycococreatortools to convert the binary masks to the polygon formatting required for COCO format and to create the JSON annotation file.

COCO, or Common Objects in Context, is an open source object recognition dataset that contains hundreds of thousands of images with millions of objects already labeled. Since it is widely-recognized, the format used to store the annotations is well supported by many deep learning implementations. I chose to use this format for my puzzle piece dataset because it is easily interpreted by the deep learning implementation I chose, so no loading customization was required.

Since more data is always better when training a deep learning model, I decided to augment my dataset by creating a set of puzzle piece images with transparent backgrounds and generating a new set of images by pasting each piece in a random location on each of 17 new backgrounds. I also created the corresponding mask for each image at the same time. I ended up creating more than 1,000 pieces and used 17 background patterns and as a result I produced more than 18,000 new images and masks.

My dataset augmentation greatly increased the number of images of puzzle pieces I had available to train my model.

Train model to locate puzzle pieces in an image

Deep learning models are best trained on machines with GPUs with lots of RAM, so I looked for a cost-effective way to use a good GPU. I found Paperspace Gradient, which did not require a credit card to sign up for the free tier, and had a few free GPU configurations available out of the box. Other options available for compute include colab and sagemaker.

With my free GPU ready, first I tried using the Tensorflow implementation of Mask R-CNN (https://github.com/matterport/Mask_RCNN) but I could only make it work on an older version of Tensorflow and training on even a small batch of images took a very, very long time. Because the compute from Paperspace Gradient that I used was free, it was limited and auto-shutdown after 6 hours, so I was not able to finish and save my results with this implementation. Thankfully, I discovered that there was a more up-to-date implementation of Mask R-CNN available in Detectron2 which uses PyTorch. Detectron2 is a modular object detection library from Facebook (https://github.com/facebookresearch/detectron2). Detectron2 cut the training time down significantly, and I was able to use my augmented dataset to train a reasonably effective model in less than an hour, well within the 6 hour free limit on Paperspace Gradient.

The inference for detecting puzzle pieces in new images based on the model weights trained with my augmented dataset is performed as one of the processing steps in the final product.

Locate corners and orient pieces

Now that I had a way to reliably find the location of each puzzle piece in a photo, I needed to tackle the problem of determining likely matches between these pieces and the puzzle in progress. First, I worked on locating the corners of each piece. The corner locations make it possible to determine the four distinct sides of each piece and make it easier to match the contours. I used the function approxPolyDP in OpenCV (Ramer–Douglas–Peucker algorithm).

Corners detected with Ramer–Douglas–Peucker algorithm

I also worked on re-orienting the pieces that were on an angle. For standard puzzles, once an image is oriented horizontally or vertically, only four possible image orientations need to be checked when trying to match the image content of a piece. I used principal component analysis to find the first two principal components and then re-oriented the piece using this information.

Various matching strategies

I then brainstormed some matching strategies on my own and with my supervisor Dr WonSook Lee. One strategy, matching based on image, divides up the full puzzle image and the puzzle in progress into grids and searches within each grid section not present in the puzzle in progress for a match among the blocking puzzle pieces using template matching.

Another strategy considered matches based on the shape of the contours of the puzzle in progress and the blockers. For puzzles that are all one colour, or very limited in colour, like the silver crypt puzzle and the gradient puzzle mentioned at the start – contour matching is really the only viable option. The implementation of contour matching remains as future work.

Sample contours highlighted

It's likely that a combination of the template matching and contour matching strategies would yield very effective results.

Communicate results of matching

As a final step, I needed to find a way to communicate the possible matches found. I decided that colour coding the edges that matched the blockers and the puzzle in progress image would be a good way to highlight the matches, as shown in the simulated illustrations below. The implementation of this results communication method remains as future work.

Illustration of colour-coded communication of matches

It would also be good to allow the user to choose to ask for only a small number of matches, if they wanted to avoid too many spoilers.

Results

Here are some inference results from running the Mask R-CNN model trained to recognize puzzle pieces. The coloured masks provide reasonably good coverage for each piece.

Inference results from Mask R-CNN model

Further inference results from Mask R-CNN model

And here are some results from template matching. Most pieces were matched correctly, except for pieces E and F. The matching is done using a grayscale version of the images and the section of the piece used for matching piece F contains no distinguishing features. In addition, template matching does not take into account the contours of the pieces, so it is not surprising that piece F was mismatched. Piece E was mismatched because it also contained a star of the same shape and the colour was not compared.

Proposed matches shown on whole image, puzzle in progress, pieces remaining

Conclusion

To conclude, I learned a lot about image manipulation and machine learning while working on this project and I am still in awe of the amount of free resources available, including libraries, free compute, and tutorials for how to accomplish almost anything.

During the course of my project, the focus shifted from image matching to concentrate more heavily on creating a model to recognize puzzle pieces. I believe the experience I have gained working on this project will serve me well in the future, whether in the workforce or in graduate studies.

Future Work

Future work could include:

- gathering more images of puzzle pieces to improve the accuracy of the piece-detection module
- refinement of the matching strategies to allow for more accurate matches including:
    - using endpiece, corner piece, and regular piece distinctions
    - using corner detection to determine the best rotation for pieces
    - using contour detection
- developing the code to colour-code the contour edges of the puzzle in progress and the corresponding matching pieces
- getting user feedback on how to communicate match results more intuitively

References and Tools Used

OpenCV https://opencv.org/ & https://docs.opencv.org/master/

Label Studio https://labelstud.io/

Mask R-CNN https://github.com/matterport/Mask_RCNN

Detectron2 https://github.com/facebookresearch/detectron2

Paperspace Gradient https://gradient.paperspace.com/

fast.ai Practical Deep Learning for Coders https://course.fast.ai/

Stanford CS231n: Convolutional Neural Networks for Visual Recognition
http://cs231n.stanford.edu/

pycococreatortools https://github.com/waspinator/pycococreator

Appendix - Code File Index

Filename	Description
create_cocofiles.py	creates the individual pieces with transparent backgrounds and the masks based on the PascalVOC format bounding boxes
augmentdata.py	creates the augmented set of puzzle piece images
custom_coco_data_segmnentation.ipynb	jupyter notebook used to “register” my test and train datasets and to train the Detectron2 implementation of the Mask R-CNN model
find_corners in puzzle_prediction.py	locate corners of an image can be found here
template_match in puzzle_prediction.py	template matching based on image
locate_pieces in puzzle_prediction.py	inference on images of puzzle pieces using model weights from training a Mask R-CNN model
pca_axis.py	find the principal components and re-orient the pieces