Automating Computational Image Analysis for Biological Experiments:

Detecting and Identifying Cockroaches to Investigate Their Complex Social Behavior

MCM104

Purpose

- Analyzing images from biological experiment is very laborious
  - I took 1 image / minute of illuminated food using iPhone
  - Two batches of pictures uploaded from iPhone every day
  - I had to search through 24,000 pictures
  - And, identify the few pictures that have cockroaches
- Manual image analysis took several days
- Need software for automatic analysis of images

Background

- Imaging is now an essential tool in biological experiments
- Image capture, storage now very easy using iPhone
- Image analysis software available for many applications
  - e.g. ImageJ and others for microscopy image analysis
- Software also available for observational experiments
  - e.g. Timelapse2 for camera trap surveys
- But, no off-the-shelf program for cockroach identification
- Identifying 400 images with cockroaches from 24000 total images is like searching the needle in a haystack

Design Criteria

The software should abide by the following criteria:

- Detect presence or absence of cockroaches appearing in the illuminated area with food
  - An image with a cockroach that is not identified is marked as a *missed* image
- Not be confused by non-interesting changes (e.g. movement of egg carton, lighting, etc).
  - An image without a cockroach that is identified is marked as a *false hit* image
- Output the file name of the pictures with cockroaches
- Be completely autonomous
  - Without need for manual intervention
- Be able to analyze the images accurately
  - Have only a small number of *missed* images
- Be able to analyze the images efficiently
  - Have only a small number of *false hit* images
- Be able to analyze and identify the images with cockroaches in them in less than an hour

Materials

- Anaconda 3 software for Python based programming
- OpenCV, NumPy, Matplotlib for image processing
- Sci-Kit's skimage for calculating SSIM
- Parallel-ssh & Paramiko for parallel image computations
- Computer cluster for parallel processing

References

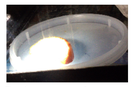
1. M. S. Norouzzadeh, A. Nguyen, M. Kosmala, A. Swanson, M. S. Palmer, C. Packer, and J. Clune, "Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning," *PNAS*, June 19, 2018 115 (25) E5716-E5725.
2. S. Greenberg, and T. Godin, "A Tool Supporting the Extraction of Angling Effort Data from Remote Camera Image," *Fisheries Magazine*, 40(6):276-287. American Fisheries Society, June 2015.
3. F-F. Yin, et al., "Computerized Detection of Masses in Digital Mammograms: Analysis of Bilateral Subtraction Images," *Medical Physics*, vol. 18, no. 5, Sept. 1991, pp. 955–963.
4. S. Yu and L. Guan, "A CAD System for the Automatic Detection of Clustered Microcalcifications in Digitized Mammogram Films," *IEEE Trans. on Medical Imaging*, vol. 19, no. 2, 2000, pp. 115-126.
5. S. Zagoruyko and Nikos Komodakis, "Learning to Compare Image Patches via Convolutional Neural Networks," *CVPR 2015*, Computer Vision Foundation.
6. S. Can, "Check If Two Images Are Equal with Opencv and Python." Pysource, 19 July 2018, pysource.com/2018/07/19/check-if-two-images-are-equal-with-opencv-and-python/.
7. K. Mark. "To Use OpenCV/cv2 to Compare and Mark the Difference between 2 Images (with Pictures)." Stack Overflow, 16 Dec. 2014, 8:34, stackoverflow.com/questions/27498833/to-use-opencv-cv2-to-compare-and-mark-the-difference-between-2-images-with-pict.
8. G. Palubinskas, "Image Similarity/Distance Measures: What Is Really behind MSE and SSIM?" *International Journal of Image and Data Fusion*, vol. 8, no. 1, 2016, pp. 32–53.
9. A. Rosebrock, "How-To: Python Compare Two Images." PyImageSearch, 11 Sept. 2018, www.pyimagesearch.com/2014/09/15/python-compare-two-images/.
10. A. Rosebrock, "Image Difference with OpenCV and Python." PyImageSearch, 5 Dec. 2018, www.pyimagesearch.com/2017/06/19/image-difference-with-opencv-and-python/.

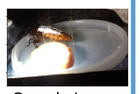
Training Procedure

Training the computer to recognize pictures with cockroaches

21 batches of images are used for training (total 11632 images)

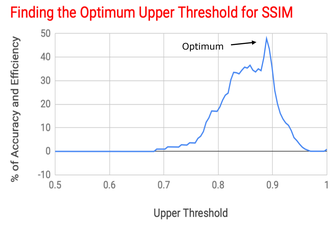
- **Classifiers for training**
  - **Classifier 1: Subtraction of images**
    - Read & convert all images into matrix of grayscale pixels using OpenCV's imread() function
    - Calculate median image from all the images in a batch using NumPy's median() function
    - Subtract every image from median image resulting in subtracted image using OpenCV's subtract()


Reference image

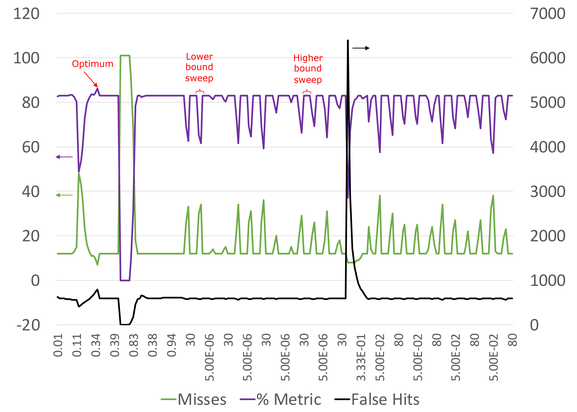

Sample image


Subtracted image

    - Segment subtracted image into 25 segments
    - Average all of the pixel values in each segment using OpenCV's mean() function
      - Each subtracted image is made up of 25 segments with a single value for each segment
    - Each segment has an upper and lower threshold value
      - The upper threshold value determines if a substantial change took place in the segment
      - The lower threshold value is a number below which the difference is assumed to be 0
    - If the segment values are within the two threshold values the image is considered to have a cockroach
    - Images that have all segment values outside of the two thresholds are considered not to have a cockroach
  - **Classifier 2: Structural SIMilarity (SSIM) Index**
    - SSIM is a scikit-image (skimage) library command that finds similarities between two images
      - SSIM compares two images and outputs a value between 0 and 1
      - The higher the value the more the similarity between the two images
    - SSIM is used to compare each image with
      a) Every image with the first image in its batch
      b) Every image with its previous image
      c) Every image with the median image of its batch
    - The SSIM values are compared to an upper and lower threshold value
    - If the SSIM values lie between the two threshold values the image is considered to have a cockroach
    - If the SSIM values are outside the range of threshold values then it is considered not to have a cockroach
- **Training Procedure**
  - The goal of training ⟶ Identify optimum values of thresholds for each classifier
  - The following metric is used to determine how good a pair of thresholds:
    - **Metric = 1 - (# misses/len(golden_truth)) - (# false_hits/len(total_images))**
      - **golden_truth**: A set of manually identified images with cockroaches in them that I made last year
        - Images determined to have a cockroach in them are stored in the *proposed* set
        - Images determined not to have a cockroach in them are stored in the *removed* set
      - The proposed set is compared to the golden_truth set to determine the number of
        - **Misses**: images that were classified as not having a cockroach but were members of golden_truth
        - **False Hits**: images that were classified as having a cockroach but were not in the golden_truth
  - I wrote a program that sweeps each threshold between two bounds using NumPy's linspace() to find the best threshold
  - Thresholds are swept one at a time
    - The metric is calculated for each threshold in the range and plotted
    - Optimum values of the threshold are determined from the plots
  - Each threshold is set to the value that leads to an optimum and the sweeps are repeated
    - If the optimum changes, the threshold value is updated
  - By iteratively updating the thresholds and repeating the sweeps an overall set of optimum thresholds can be determined



Training Results



- Graph combining sweeps of all of the classifiers with the misses and false hits
- You can see that whenever the false hits or misses are high, the percent metric drops
- Optimum training metric is presented in the table below

Validation Procedure & Results

- 18 batches of images used for validation (12167 images)
  - The optimum classifier thresholds are applied to classify the validation set
  - The resulting metric is presented below

	Training	Validation		
Classifier	Metric	Metric	Misses	False Hits
Median SSIM	45.2%	25.4%	28	8086
Previous image SSIM	47.9%	34.1%	132	3282
First image SSIM	2.3%	4.4%	1	11618
25 Segments	85.9%	68.2%	72	1287
All together	86.2%	73.7%	54	1266

- The table shows, combining multiple classifiers increases the accuracy and efficiency of the program
- If one of the classifiers misses an image with a cockroach in it, another one will find it

Conclusions

- Accuracy and Efficiency increases with number of classifiers
- When multiple classifiers are combined they make up for each others mistakes
- Adding more classifiers is likely to lead to even better performance

Future Work

- Neural Networks offer the potential for mimicking extremely large numbers of classifiers
  - Next year I will use neural networks which will increase the efficiency and accuracy close to 100%
- I will extend this work to vision based observation problems in biological sciences, physical sciences, medical sciences, and robotics