Computational Cognitive Science
Assignment 2 - Report

ASSIGNMENT - 2

March 8, 2019

Students:
Pawan Agarwal (15817482)
Dheeraj Athrey (150231)
Pranav Sao (150504)

Instructions

Instructions to run the code are given in the README file.

Implementation

Q1

The goal of the question is to design complex cells that can detect a square or a triangle using a bank of orientation-selective 2D Gabor filters.

We chose two shapes, hollow triangles and squares. We chose five images to classify, blue and red squares and triangles and a background. Here background means no shape. We converted the images into a color independent form using a conversion formula. Since the gabor filters are being used on these modified images(color independent) to detect the shape, the color of the input shape shouldn't affect the results. The images were all resized to 72x72 to suit Q2.

Now coming to the shape detection, initially we applied gabor filters on the shapes and tried applying corner detection using external libraries like Harris Corner Detector and then distinguish the shapes based on the number of corners. But later we realized that this not the correct way as the problem can also be solved without a gabor filter using this approach.

Later, we used only gabor filters and some mathematical logic to find the number of corners in the shape whcih was used as a metric to decide whether an image is a square or a triangle or a background. The logic used to detect corners of the shapes is simple. A Gabor filter responds to edges and texture changes. For example, in the case of a

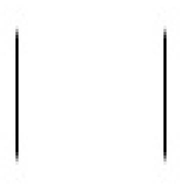

Figure 1: Vertical edges

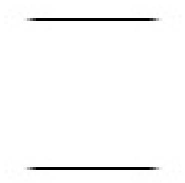

Figure 2: Horizontal edges

Figure 3: Outputs when gabor filters are used on a square

square, we used it to detect the edges. We used two gabor filters one to find the horizontal edges and one for the vertical edges. The parameters of these filters were tuned to get a precise output from the filter. Now coming to the mathematical logic, if we apply bitwise negation on these outputs we have high values only at places where the filter has detected an edge. Multiplying these two outputs would give us the points which are corners.

Similar logic can be used for a triangle. We find the three edges using proper tuning of gabor filters. And using these edge outputs in pairs we can find the intersection points by using same logic as with the square.

Q2

Template Generation for the experiment

A template of size 720x720 was generated where each frame is of size 72 x 72 and can hold one shape. So a total of 100 shapes can be used. We represent each square frame with the coordinates of the top left corner. Now based on the input number of frames, we randomly choose that many frames from these 100 frames and insert the shapes according to the type of experiment chosen, i.e. Feature search or Conjunction search. We remember the frames where shapes were put while generating the template.

Explanation

For feature search, the distinguishing factor is the shape. Now we make a loop through all the remembered shapes' frames in the template and then for each frame, we detect if there is a square or a triangle using the solution developed in Q1.

Now coming to the conjunction search, we have features both shape and color. To detect the shape we used the same logic as used in the feature search, i.e. using the logic from Q1. But for detecting the colour we used thresholding to distinguish the images. We converted the image to a grayscale and applied binary thresholding on it and based on the values of the image matrix, we figured out what the colour of the image was. We make a loop randomly through all the saved frames and apply the aforementioned logic to deduce both their shapes and colours.

While running the code with proper parameters all the shapes and colors of frames are recognized and their co-ordinates along with their

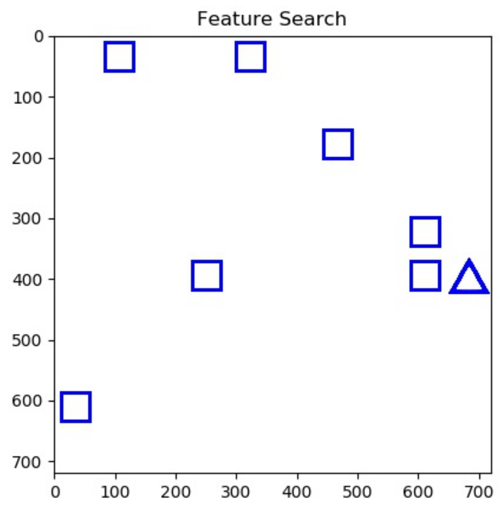

Figure 4: Template for feature search with 8 objects

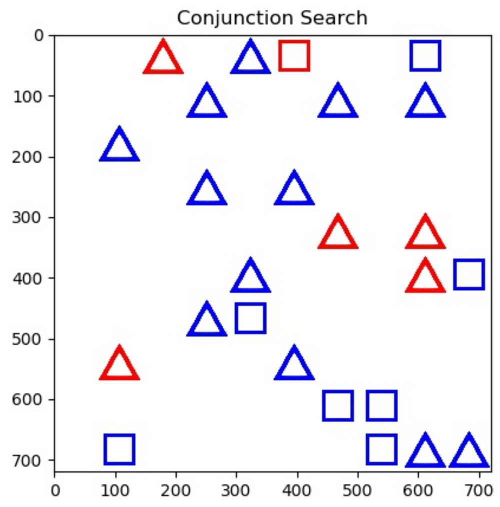

Figure 5: Template for conjunction search with 25 objects

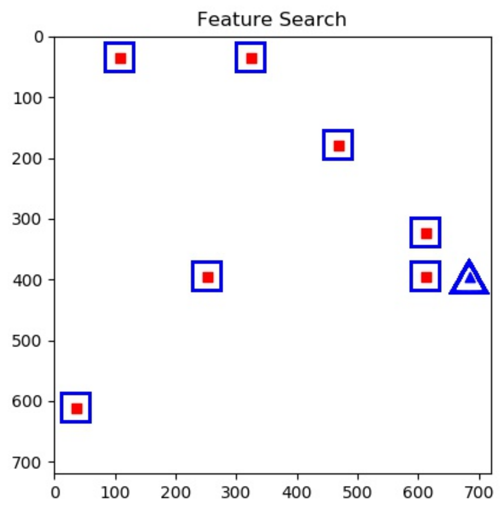

Figure 6: Feature Search with 8 objects after running the Q2 algorithm

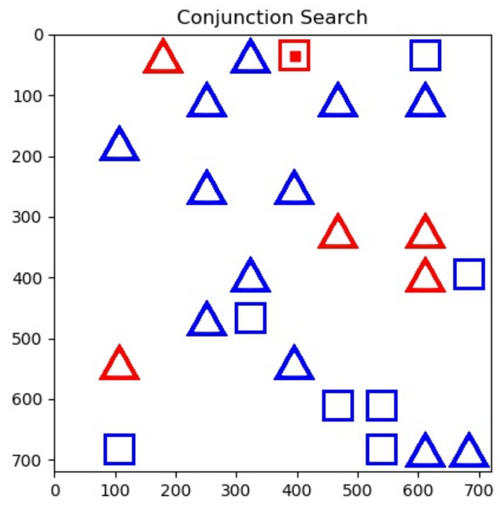

Figure 7: Conjunction Search with 25 objects after running the Q2 algorithm, dot in the middle represents the odd stimulus

shapes and colors are printed on the terminal.

Q3

In Q3 we need to compare the times taken by feature search and conjunction search.

Using the logic from Q2, for various numbers of objects, we plotted graphs for time taken for feature search and conjunction search vs the number of objects. The search stops whenever we reach the target odd stimulus. We calculated the time taken for each choice of search, for each choice of number of objects, 20 times and considered the average time taken.

The results are as follows:

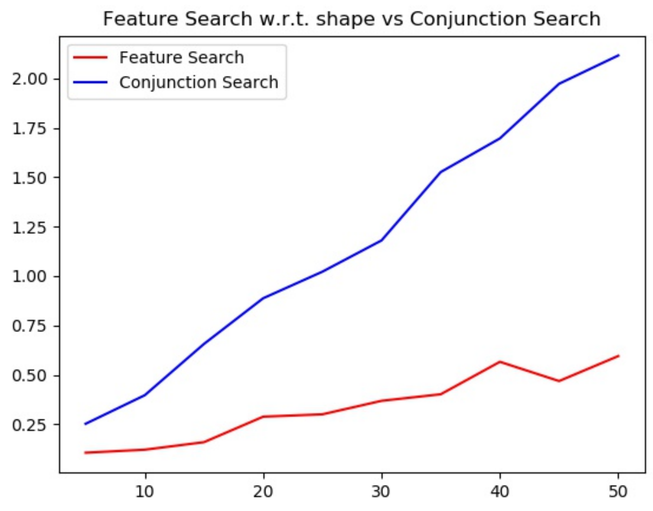

Figure 8: Feature Search w.r.t. shape vs Conjunction Search

In the above figure, as you can observe, the time taken for conjunction search is increasing with the increase in number of objects much faster than it did for feature search. This is because in the case of conjunction search, we are making color map and shape map parallely and then combining them serially which takes a longer time compared to feature search which directly searches in a map.

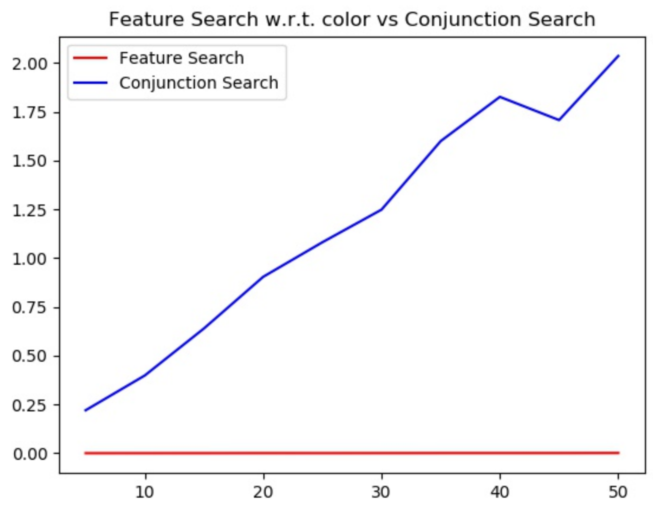

Figure 9: Feature Search w.r.t. color vs Conjunction Search

In the above figure, the time taken for feature search is almost the same even with higher number of objects because, it's a feature search based on color as the pop out factor and not shape. Ass it's solved using a color thresholding and hence is very fast even for large number of objects.

Note

If there's any problem in understanding any part of the code or the report , please email us at dheerajb@iitk.ac.in or raise a issue on github. `https://github.com/pawan47/feature_and_conjunction_search`