Image Analysis Assignment 3

Olof Harrysson

1.

I first thought that the images differed by a shift in both x and y. I therefore tried to shift one of the images back and thereby easily see the difference between them by plotting the abs(img1 - img2). I tried to find this bit shift by padding one of the images with zeros and finding at what coordinate it best matched with the other image with the function normxcorr2, a technique called template matching. From there I could crop out the unshifted image from the padded image.

This didn't give me the expected result and I suspect it's because the image is not shifted but instead scaled in size.

I therefore identified SIFT feature points to eventually fit one image to another. From there I followed my initial approach of plotting the abs difference. This showed me that there still were some edges present so I eroded the image. Finally I tried to create bounding boxes around the object but wasn't successful when I tried to merge overlapping boxes. I didn't write all the code myself but found much of the code online.

Image 1

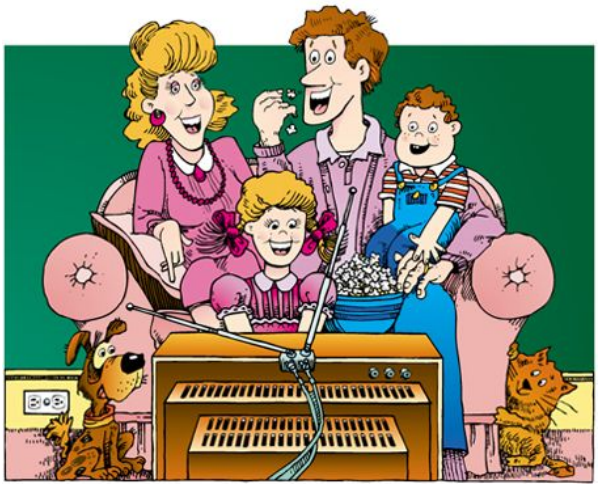
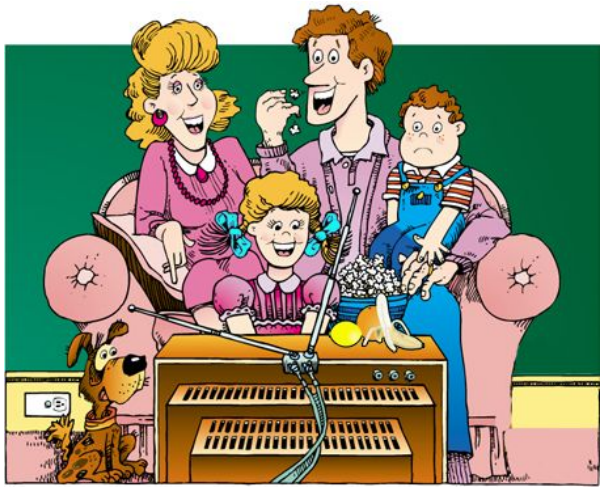
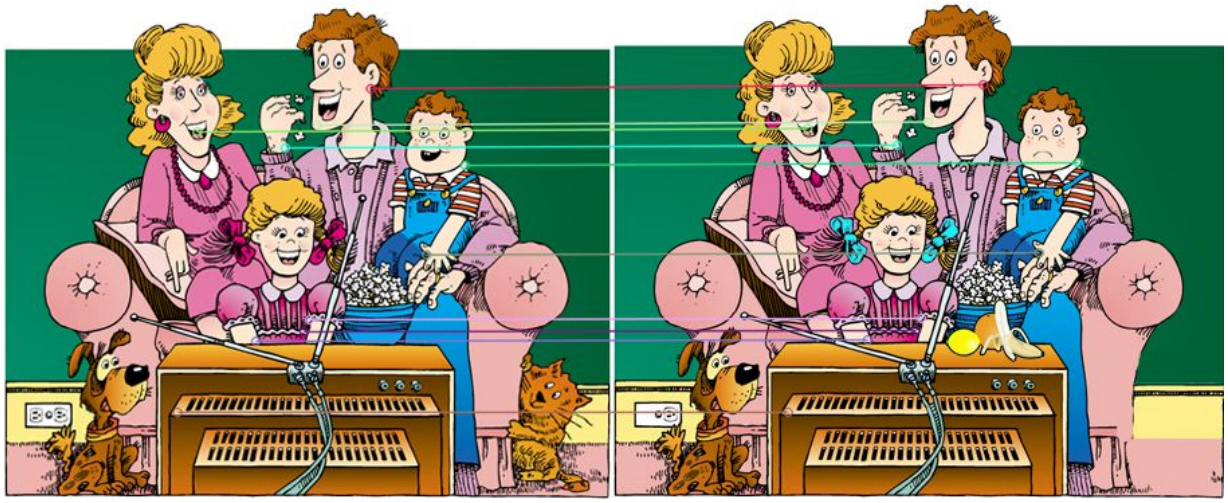


Image 2



Strongest Matching Feature Points



Abs diff

Bounding boxes

2.
Chamber: mu = 0.35421, sigma = 0.099198
Background: mu = 0.106461, sigma = 0.0935997

When the prior is set to low parts of the background is included in the segmentation. If you set it to high the segmented picture will be very small. There was no difference between a prior of 0.42 and 0.5.

Prior = 0.4 	Prior = 0.42 
Prior = 0.5 	Prior = 0.6 

3.

```
import numpy as np

p1 = [[3, 2, 1, 0], [2, 2, 2, 0], [2, 1, 2, 1]]
p2 = [[1, 2, 2, 3], [1, 1, 0, 2], [3, 1, 2, 0]]

F = np.array([[-4, 2, -6], [3, 0, 7], [-6, 9, 1]])

a = np.array([[1,2], [3,2], [0,3]])
b = np.array([[1,1], [5,1], [-1, -3]])


correponding_points = []
for ai in a:
  x1 = np.append(ai, 1)

  for bi in b:
    x2 = np.append(bi, 1)[np.newaxis] # Add axis to transpose

    if x1.dot(F).dot(x2.T) == 0:
      correponding_points.append((ai, bi))
```

```
for point1, point2 in correponding_points:
  print("point {} corresponds to {}".format(point1, point2))
```

Gives the output

```
point [1 2] corresponds to [5 1]
point [3 2] corresponds to [1 1]
```

4.

Prior to changing anything I measured the hitrates for the datasets shown in the table below. It didn't perform well at all on the home datasets. Upon further inspection I noticed that the segmentation didn't work as intended. Since there were noise in the datasets it often segmented out noisy pixels. To fix this I changed the im2segment to pick the five biggest segments.

Dataset	Hitrate
Short1	0.9800
Short2	0.9800
Home1	0.0360
Home2	0.0420
Home3	0.0470

Old system

The classifier worked well for the short datasets but poorly for the home datasets. Comparing these datasets I could see that the home datasets were noisier and scaled differently. The features I had picked in the previous assignment weren't robust enough for this kind of data. For example, one of my features were the number of feet a letter had. It's still a sound idea but the implementation wasn't robust enough. It looked from the bottom of the image until it found a row with values and checked how many segments this row had. If a noisy pixel appeared under the real letter this implementation wouldn't be able to handle it.

Features

To handle this and other problems I changed my features. Many of the features was not calculated on the whole image but a cropped out image of the segment.

First group of my features was looking at how bright different subimages were. By bright I mean the number of white pixels divided by total pixels. I looked at the image as a whole, divided the

image into halves (top, bottom, left, right) and quadrants (topleft, topright, bottomleft, bottomright).

My other features were roughly the perimeter of the letter, the center of mass coordinates, number of segments on the inverted image, sum of edges in the x-axis, number of feet the letter has and width to height ratio.

I also experimented with different features but wasn't successful with them. One such feature would roughly be the coordinates for the strongest sift feature point.

Classifiers

Another big part of my process was that I tried different classifiers. I tried nearest neighbour, support vector machine, decision trees and ensemble methods. Nearest neighbour outperformed the other techniques which surprised me. Perhaps some of the other methods needed more data to give better result.

I tweaked the settings for the nearest neighbour and found that most fancy settings didn't give me a performance boost. One important setting was to normalizing the features to avoid one feature dictate the whole outcome.

Possible improvements

As the training data is pretty small I think expanding it artificially would increase the system's performance. For example, one could take every image and apply transformations to it to get a slightly different image. You could also add some images with salt and pepper noise, but instead of black and white, more like light gray and dark gray.

Another problem with the training data is that there is only one occurrence of some letters. By expanding it it's possible to run the nearest neighbour algorithm that looks at the k-nearest neighbours instead of the 1-nearest neighbour it's now using.

My final results are shown below with a mean of 0.8642

Dataset	Hitrate
Short1	1
Short2	0.9600
Home1	0.7700
Home2	0.7980
Home3	0.7930