Homework Assignment 9
Computer Vision for HCI
Prof. Jim Davis
TA: Sayan Mandal

Anirudh Ganesh
CSE5524 (Au '18)
Score: ___/12
Due Date: 10/30/18

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from skimage.io import imread
        from scipy.signal import fftconvolve
        from sklearn.neighbors import KNeighborsClassifier
        from matplotlib.colors import ListedColormap
```

1 Part 1 Template Matching NCC

```
In [2]: search = imread('./data/search.png')
        template = imread('./data/template.png')
```

```
In [3]: image_shape = search.shape

        image = np.array(search, dtype=np.float64, copy=False)
```

```
In [4]: pad_width = tuple((width, width) for width in template.shape)
```

```
In [5]: image = np.pad(image, pad_width=pad_width, mode='constant', constant_values=0)
```

```
In [6]: def calc_window_sum(image, window_shape):
            window_sum = np.cumsum(image, axis=0)
            window_sum = (window_sum[window_shape[0]:-1] - window_sum[:-window_shape[0] - 1])
            window_sum = np.cumsum(window_sum, axis=1)
            window_sum = (window_sum[:, window_shape[1]:-1] - window_sum[:, :-window_shape[1] -
            window_sum = np.cumsum(window_sum, axis=2)
            window_sum = (window_sum[:, :, window_shape[2]:-1] - window_sum[:, :, :-window_shape
            return window_sum
```

```
In [7]: image_window_sum = calc_window_sum(image, template.shape)
        image_window_sum2 = calc_window_sum(image ** 2, template.shape)
```

```
In [8]: template_mean = template.mean()
        template_volume = np.prod(template.shape)
        template_ssd = np.sum((template - template_mean) ** 2)
```

```
In [9]: xcorr = fftconvolve(image, template[::-1, ::-1, ::-1], mode="valid")[1:-1, 1:-1, 1:-1]
```

```
In [10]: numerator = xcorr - image_window_sum * template_mean
```

```
In [11]: denominator = image_window_sum2
```

```
In [12]: image_window_sum = np.multiply(image_window_sum, image_window_sum)
```

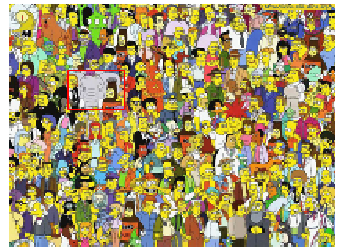
```
In [13]: image_window_sum = np.divide(image_window_sum, template_volume)
```

```
In [14]: denominator -= image_window_sum
         denominator *= template_ssd

In [15]: denominator = np.maximum(denominator, 0) # sqrt of negative number not allowed

In [16]: denominator = np.sqrt(denominator)

In [17]: response = np.zeros_like(xcorr, dtype=np.float64)

In [18]: mask = denominator > np.finfo(np.float64).eps

In [19]: response[mask] = numerator[mask] / denominator[mask]

In [20]: slices = []

In [21]: for i in range(template.ndim):
             d0 = template.shape[i] - 1
             d1 = d0 + image_shape[i] - template.shape[i] + 1
             slices.append(slice(d0, d1))

In [22]: result = response[tuple(slices)]

In [23]: def largest_indices(ary, n):
             """Returns the n largest indices from a numpy array."""
             flat = ary.flatten()
             indices = np.argpartition(flat, -n)[-n:]
             indices = indices[np.argsort(-flat[indices])]
             return np.unravel_index(indices, ary.shape)

In [24]: ind_list = [1, 2, 5, 10, 100, 500]
         x_list = []
         y_list = []

         for t in ind_list:
             ind = largest_indices(result, t)
             x_t, y_t = ind[1][-1], ind[0][-1]
             x_list.append(x_t)
             y_list.append(y_t)

In [25]: ij = np.unravel_index(np.argmax(result), result.shape)
         _, x, y = ij[::-1] # IMPORTANT TO SWITCH ROWS AND COLUMUNS I've LOST 5 points this seme

In [26]: plot_n = 6

         fig, axarr = plt.subplots(6, 2, figsize=(15,20))

         for i in range(plot_n):
             htemplate, wtemplate, _ = template.shape

             axarr[i, 0].imshow(search[y_list[i]: y_list[i]+htemplate, x_list[i]:x_list[i]+wtemp
```

```
    axarr[i, 0].set_axis_off()
    axarr[i, 0].set_title(f'detected region, {ind_list[i]} closest')

    axarr[i, 1].imshow(search)
    axarr[i, 1].set_axis_off()
    axarr[i, 1].set_title(f'image: ({x_list[i]},{y_list[i]})')


    rect = plt.Rectangle((x_list[i], y_list[i]), wtemplate, htemplate, edgecolor='r', f

    axarr[i, 1].add_patch(rect)

plt.show()
```

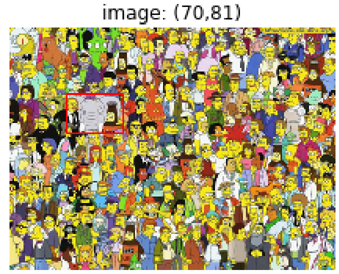
detected region, 1 closest

image: (71,82)



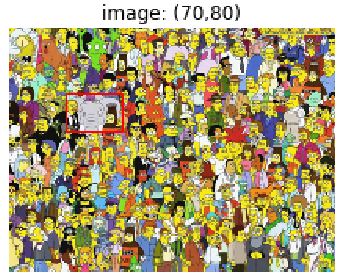
detected region, 2 closest

image: (70,81)



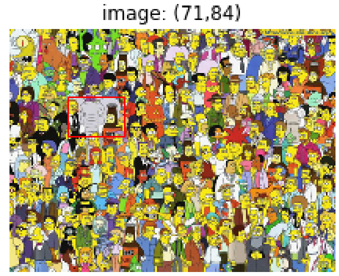
detected region, 5 closest

image: (70,80)



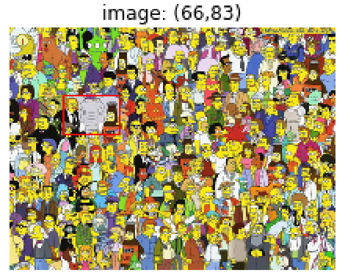
detected region, 10 closest

image: (71,84)



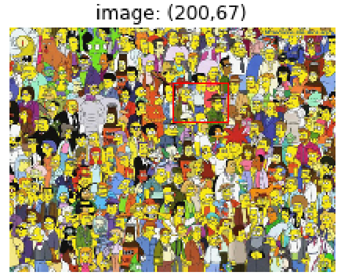
detected region, 100 closest

image: (66,83)



detected region, 500 closest

image: (200,67)



We can see that our template matching performs really well in that it identifies "Stampy" the elephant pretty much pixel for pixel. Even the 2nd and 5th closest guesses are almost indistinguishable because of their 1-2 pixel discrepancy. We see however that in 100th closest our match drifts little too much to the left, as witnessed by the fact that there is more of John Swartzwelder (the guy in suit) and Brandine (the lady on the right) is now nowhere to be seen. By 500th closest, we can clearly see that the matching is way off, (its now closer to Homer rather than Stampy). This is in line with what we discussed in the slides with the person's image and the mountain behind

2 Part 2 Template Matching Stereo Vision

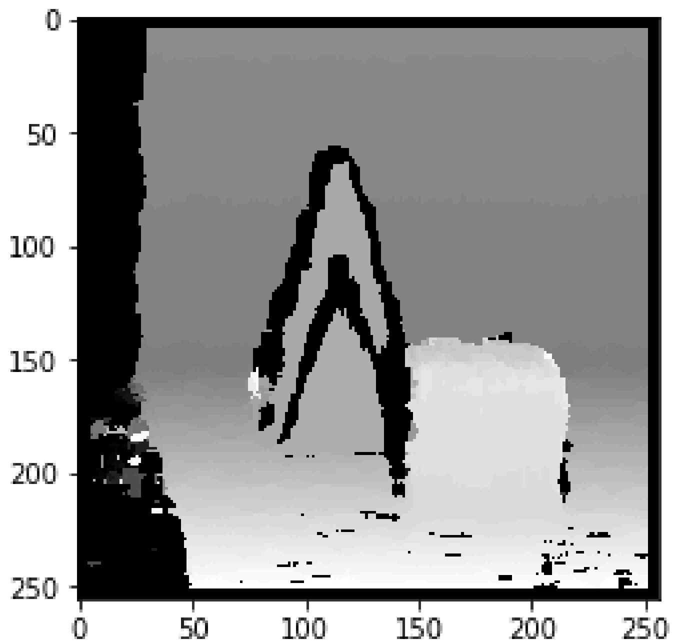
```
In [27]: left = imread('./data/left.png')
         right = imread('./data/right.png')
```

```
In [28]: def stereo_match(left_img, right_img, kernel=11, max_offset=50):

             w, h = left_img.shape
             depth = np.zeros((w, h), np.uint8)
             depth.shape = h, w
             kernel_half = kernel // 2
             offset_adjust = 255 / max_offset

             for y in range(kernel_half, h - kernel_half):

                 for x in range(kernel_half, w - kernel_half):
                     best_offset = 0
                     prev_ncc = float("-inf")

                     for offset in range(max_offset):

                         ncc = 0

                         lwindow = left[y-kernel_half:y+kernel_half, x-kernel_half:x+kernel_half
                         rwindow = right[y-kernel_half:y+kernel_half, x-kernel_half:x+kernel_hal

                         lwindow_mean = lwindow.mean()
                         lwindow_std = np.std(lwindow)
                         rwindow_mean = rwindow.mean()
                         rwindow_std = np.std(rwindow)

                         for v in range(-kernel_half, kernel_half):
                             for u in range(-kernel_half, kernel_half):
                                 ncc_temp = (int(left[y+v, x+u]) - lwindow_mean) * (int(right[y+
                                 if lwindow_std ==0 or rwindow_std ==0:
                                     ncc_temp = 0
                                 else:
```

```
                    ncc_temp /= lwindow_std*rwindow_std
                ncc += ncc_temp


            if ncc > prev_ncc:
                prev_ncc = ncc
                best_offset = offset

        depth[y, x] = best_offset * offset_adjust

    return depth
```

In [30]:
```
plt.imshow(stereo_match(left, right), cmap='gray')
```

Out[30]:
```
<matplotlib.image.AxesImage at 0x1cb88eaffd0>
```

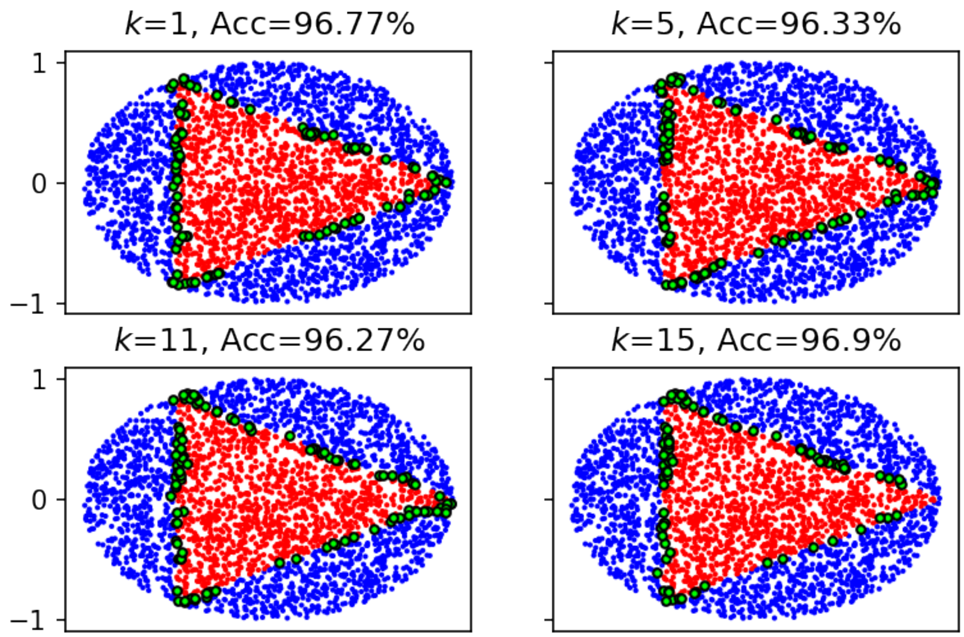


3 Part 3 k-Nearest-Neighbours

In [31]:
```
train = np.loadtxt('./data/train.txt')
test = np.loadtxt('./data/test.txt')
X_train, y_train = train[:,0:2], train[:,2]
X_test, y_test = test[:,0:2], test[:,2]
```

In [32]:
```
neighbours = [1, 5, 11, 15]
```

```
In [33]: f, axarr = plt.subplots(2, 2, sharex='col', sharey='row', dpi=150)
         for n in neighbours:
             idx = neighbours.index(n)
             model = KNeighborsClassifier(n_neighbors=n)
             model.fit(X_train,y_train)
             y_pred = model.predict(X_test)
             cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
             axarr[idx//2,idx%2].tick_params(axis='x', which='both', bottom=False, top=False, la
             axarr[idx//2,idx%2].scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap=cmap_bold, s
             axarr[idx//2,idx%2].scatter(X_test[y_pred!=y_test, 0], X_test[y_pred!=y_test, 1], c
             axarr[idx//2,idx%2].set_title(f"$k$={n}, Acc={round(model.score(X_test, y_test)*100
         plt.show()
```



We notice that most of the errors made in classification, it is at the border of our classes, this is expected as that is where the nearest neighbours will be the most contested. However as we increase our k, at first we notice that our accuracy decreases, as in this case there might be cases where more neighbours cause the classifier to be misled. But once we reach k=15, we see that our accuracy jumps up, thus indicating that at this stage we are getting sufficient enough information from our neighbours that its indicative of our overall sample.