Assignment 1

Tanuj Kumar kumarta3 1002197133

September 2017

1 Question 1

1.1 1 a

Let $\delta(t)$ be the one dimensional impulse function at t. Define a 2D space with axes x and y.

Then, we can define the **two-dimensional impulse function** as:

$$\delta(u,v) = \begin{cases} 1 & (u,v) = (0,0) \\ 0 & \text{else} \end{cases}$$

Where u and v are spatial two-dimensional coordinates on the x and y axes respectively. If we were to define two separate 1D impulse functions $\delta_x(u)$ and $\delta_y(v)$, we could also write our 2D impulse function as a product of these:

$$\delta'(u,v) = \delta_x(u)\delta_y(v)$$

1.2 1 b

Consider a 2D impulse located at coordinates $u = m$ and $v = n$. Our 2D impulse function at position (m,n) is simply:

$$\delta_{(m,n)}(u,v) = \delta'(u-m, v-n) = \delta_x(u-m)\delta_y(v-n)$$

We can show this is the case: $\delta_{(m,n)}(m,n) = \delta'(m-m, n-n) = \delta(0,0) = 1$, as required.

1.3 1 c

Python code:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D, get_test_data
from matplotlib import cm
import numpy as np
```

```
def pulse(m, n, tups):
   img = np.zeros(shape=(m,n))
   for t in tups:
      img[t[0]-1][t[1]-1] = 1
   return img

pops =
    [(10,20),(20,40),(30,60),(40,80),(50,100),(60,80),(70,60),(80,40),(90,20)]
impulseimg = pulse(100, 200, pops)

fig = plt.figure(figsize=plt.figaspect(0.5))

ax = fig.add_subplot(1, 2, 1, projection='3d')

X = np.arange(1,201,1)
Y = np.arange(1,101,1)

X, Y = np.meshgrid(X, Y)

surf = ax.plot_surface(X, Y, impulseimg, rstride=1, cstride=1,
    cmap=cm.coolwarm, linewidth=0, antialiased=False)

ax.set_zlim(-0.01, 1.01)
fig.colorbar(surf, shrink=0.5, aspect=10)

plt.show()
```

Plot will be below, captioned.

1.4 1 d

In two dimensions, we define $f(u, v)$ to be some 2-dimensional function that we want to produce as a combination of 2-dimensional impulses. Again we use our defined 2D impulse function $\delta(u, v)$. Using a similar idea to the 1D function sum, we can write:

$$f(u,v) = \sum_{j=1}^{N} \sum_{i=1}^{M} f(u_j, v_i)\delta'(u - u_j, v - v_i)$$

Where we have a $M \times N$ image, and j represents the horizontal coordinate index, while i represents the vertical coordinate index. This is equivalent to scaling each (i, j) position in the image by $f(u_j, v_i)$ (provided that there is an impulse here i.e. this part of the image is not simply black). Thus altogether we get a 2D function $f(u, v)$ represented and scaled by impulses.

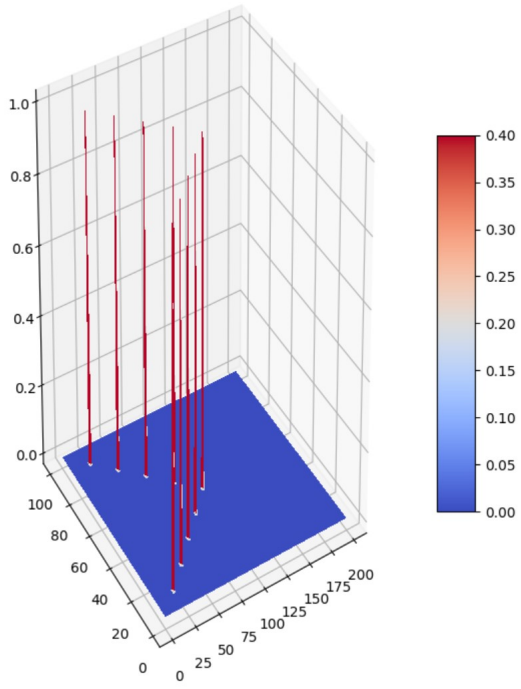

Figure 1: The plot of the image as described in question 1c). It forms roughly a triangular shape on the left side of the image

2 Question 2

2.1 2 a

Let $n \times n$ be the image size and $m \times m$ be the convolution filter size. In total there are n^2 pixels in the image which the filter will convolve with. The convolution involves m^2 operations total, *per pixel*. Therefore, the time complexity of base convolution is $O(n^2 m^2)$.

2.2 2 b

Let our initial $m \times m$ filter h be **separable**. Then we can separate h into a 1D vertical and 1D horizontal filter of size m each, and then only do one pass with each filter. This means instead of m^2 operations, we only do $2m$ operations per pixel. This means that our time complexity has been reduced to $O(n^2 2m) = O(mn^2)$.

2.3 2 c

We can figure out whether both filters are separable based on singular value decomposition of the matrix. If a matrix's SVD form has only one singular value, then it is separable. We will use the following Python code:

```
import numpy as np

a = np.matrix('10 40 8; 5 3 5; 12 5 2')
b = np.matrix('6 3 6; 2 1 2; 6 3 6')

U1, s1, V1 = np.linalg.svd(a, full_matrices=True)
U2, s2, V2 = np.linalg.svd(b, full_matrices=True)

print(s1)
print(s2)
```

From the above code, we see that F_1 **is not separable**, while F_2 **is separable**. The one non-zero singular value is $3\sqrt{19}$.

For F_2, the horizontal and vertical filters are as follows:

$$F_2^h = \sqrt{3}\sqrt[4]{19}\begin{bmatrix}-0.6666 & -0.3333 & -0.6666\end{bmatrix}$$

(FIX THIS)

$$F_2^v = \sqrt{3}\sqrt[4]{19}\begin{bmatrix}-0.6882472 & -0.22941573 & -0.6882472\end{bmatrix}^T$$

(FIX THIS)

3 Question 3

3.1 Rough Outline of Procedure

We scan in the templates and then loop through each template, performing a normalized cross correlation with the original image. This gives us a set of 30 different normalized cross correlations. We will call these *layers*. We then grab the pixels past a defined threshold T that are maxima among every such layer for that particular pixel, and then store their pixel coordinates along with which template they correspond to. Then for every max point candidate, it may or may not be a digit. We check a 3 x 3 box around the candidate pixel using its stored coordinates to see if it is a local maxima, and if it is, we draw a box with the label of the template it belongs to, on the original image.

3.2 Matlab Code (a - f)

3.2.1 correlation.m (as of f)

```
[templates, dims] = readInTemplates();
thermo = imread('thermometer.png');
imshow('thermometer.png');
drawnow;
thermo = rgb2gray(thermo);
thermo = double(thermo);
M = size(thermo,1);
N = size(thermo,2);
T = 0.68; % part e T was 0.5 , 0.68 is the ideal

corrArray = zeros(M, N, 30);

for( t = 1 : 30 )
    tempo = double(rgb2gray(templates{t}));
    th = dims(t).height;
    tw = dims(t).width;
    out = normxcorr2(tempo, thermo);
    % figure, surf(out), shading flat
    offsetX = round(tw/2);
    offsetY = round(th/2);
    out = out(offsetY : offsetY + M - 1, offsetX : offsetX + N - 1);
    corrArray(:,:,t) = out; % would this work -- what dims does
        normxcorr output
end

[maxCorr, maxIdx] = max(corrArray,[],3); % is maxcorr from 0 to 1?
% idea:
% maxCorr is thermo.png image-sized
% maxCorr stores the maximum pixel value for that pixel from all
    templates
% maxIdx should store the template index for it?
[candY, candX] = find(maxCorr > T);
% (candX, candY) gives the INDEX of the pixel value for a maxCorr pixel
    exceeding T
% [candX, candY] is (number of exceeding pixels) x 2 dimensions
% (candX, candY) is the INDEX of the pixel in maxCorr FOR THE IMAGE
% so to find which template its on: maxIdx(candY(i),candX(i)) where i =
    0 :
% size of [candX, candY].
% we put this in an array called templateIndex
% and then we extract the correlation matrix from templateIndex

numMaxPts = size([candY, candX], 1);
templateIndex = zeros(numMaxPts, 1);

% numMaxPts holds the number of maximum points

% [candY(1), candX(1)]
```

```
for( z = 1 : numMaxPts )
    templateIndex(z) = maxIdx(candY(z), candX(z));
end

% templateIndex:
% We iterate through the max points
% For templateIndex index z, we put the template index corresponding to
    the
% max point (candY, candX) by the index of the [candX, candY] value
% templateIndex contains numMaxPts values
% each value corresponds to a max point exceeding the threshold
% each value is the template index number that the max point was from

% thisCorr = corrArray(:,:,templateIndex); % does this work out???

% this gives us the original corrArray but
% instead of 30 dimensions, its ...

for ( x = 1 : numMaxPts )
    thisCorr = corrArray(:,:,templateIndex(x));
    % this should give us the corrArray for a specific template
    % now we have: the thisCorr image correlated with the template
    % candidate
    % we also have the coordinates of the max points:
    %(candY(x), candX(x))
    % so we have to check if it is a max in a 3 x 3 box.
    % If true, we draw a box around it (which shouldn't be bad).
    % Repeat this for every max point.
    row = candY(x);
    col = candX(x);
    if isLocalMaximum(col,row,thisCorr) == 1
        % draw bounding box
        tInd = templateIndex(x);
        drawAndLabelBox(col,row,tInd,dims)
    end
end
```

3.2.2 isLocalMaximum.m

```
function truth = isLocalMaximum(x, y, thisCorr)
    val = thisCorr(y,x);
    truth = 1;
    for (i = y-1 : y+1)
        for (j = x-1 : x+1)
            if and(thisCorr(i,j) >= val, or(i ~= y, j ~= x))
                [i,j;y,x];
                truth = 0; % not a local maximum
```

```
            end
        end
    end

end
```

3.3 Threshold Values

The initial threshold value was 0.5 (for part e), which proved to be much too loose. After testing a number of different threshold values, the ideal threshold value was **0.68**. No digits were incorrectly labelled or detected, however at 0.68 we missed every "1" and a number of other digits. Exceeding 0.68 results in more misses, while going below 0.68 (such as 0.66) results in slightly more positive detection but more incorrect labelling (such as labelling any singular segment being lit up as a "1", such as when it is part of a bigger number such as 7).

3.4 Bonus

The performance is expected to improve because cropping templates directly from the image will allow the correlation to pick up unique variations to each digit in the original image itself. For example, differences in the shadows in the image correspond to different grayscale brightness values in the new cropped templates. However, this is an instance of "using your test data as your training data" in machine learning, which is generally a form of potentially getting your model to overfit at the expense of being able to use it to reliably detect numbers elsewhere.

3.5 Detected Images

Figure 2: $T = 0.50$. Captures non-digits as digits. This was the original option for T in part 3e.

Figure 3: $T = 0.66$. Mislabels some individual digital segments as ones.

Figure 4: $T = 0.68$. Ideal threshold value. Misses all ones and some other numbers. Small test variations were made in the range 0.675 to 0.715 and they are all generally the same.

Figure 5: $T = 0.70$. Past this less digits will be detected.