Image Processing hw1 Report

Andrey Yershov

I. Problem 1

A. nearest neighbor

The first problem was attempted the last in the list, as long as I did not really know where to start when a read the task. For the nearest neighbor interpolation, I simply divided every new image's pixel by the scale input and rounded the result to the nearest integer. Two if statements were used in order to handle array exceeding errors on the first row and column. The pixels from original image were mapped to the new image using calculated coordinates.

```
clear
prompt = 'Please type in scale:';
scale = input(prompt);
image_or = imread('cat120.jpg');
[rows_or, cols_or] = size(image_or);
scaled_im = zeros(scale*rows_or, scale*cols_or);
[rows,cols]=size(scaled_im);

%nearest neighbor
for x = 1:rows
        for y = 1:cols
                a = round(x/scale);
                b = round(y/scale);
            if a<1
                a=a+1;
            end
            if b<1
                b=b+1;
            end
            scaled_im(x,y) = image_or(a,b);
        end
end
figure(1)
subplot(2,1,1), imshow(image_or), title('Original')
hold on
subplot(2,1,2), imshow(scaled_im,[]), title('NN scaled')
hfig = figure (1)
print(hfig, '-dpng', '-r300', 'NN')
```

B. Results

The results were not surprising, however somewhat hard to interpret. Scaling with factor less than 1 has decreased the number of distinct pixels in the image, thus making the details less visible. An example with scale factor of 0.4 was used.

Next, two examples with scaling factors of 2.5 and 13.11 were made. The resultant images were similar, even when scaled. I believe this was due to image viewer's built in interpolation for zoomed in images. The algorithm undoubtedly worked, as long as the number of pixels in the obtained image matrix increased proportionally to the input scale.

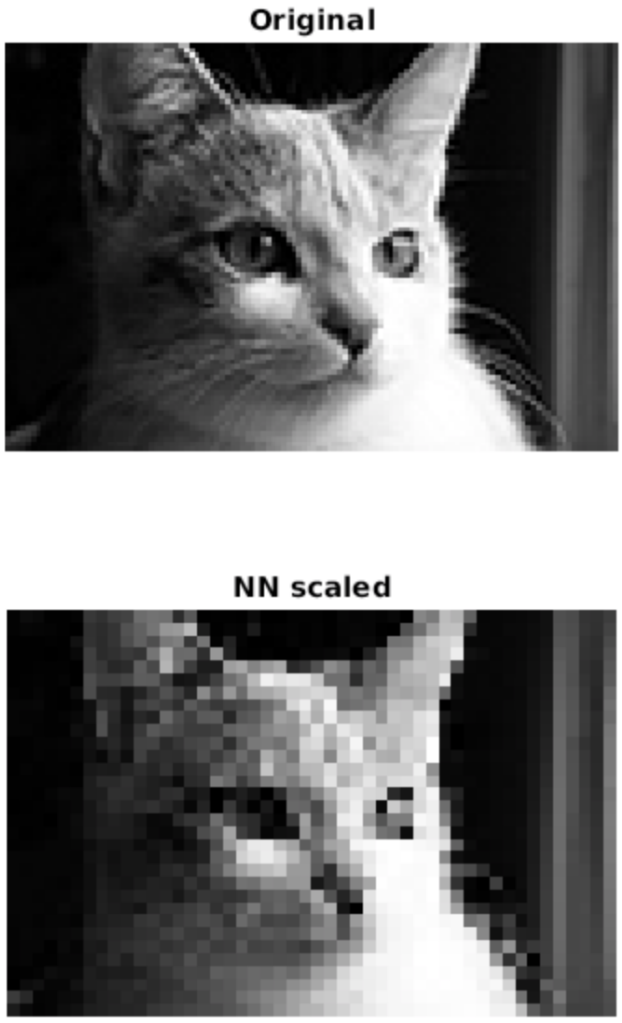

Fig. 1. Image scaled 0.4 times

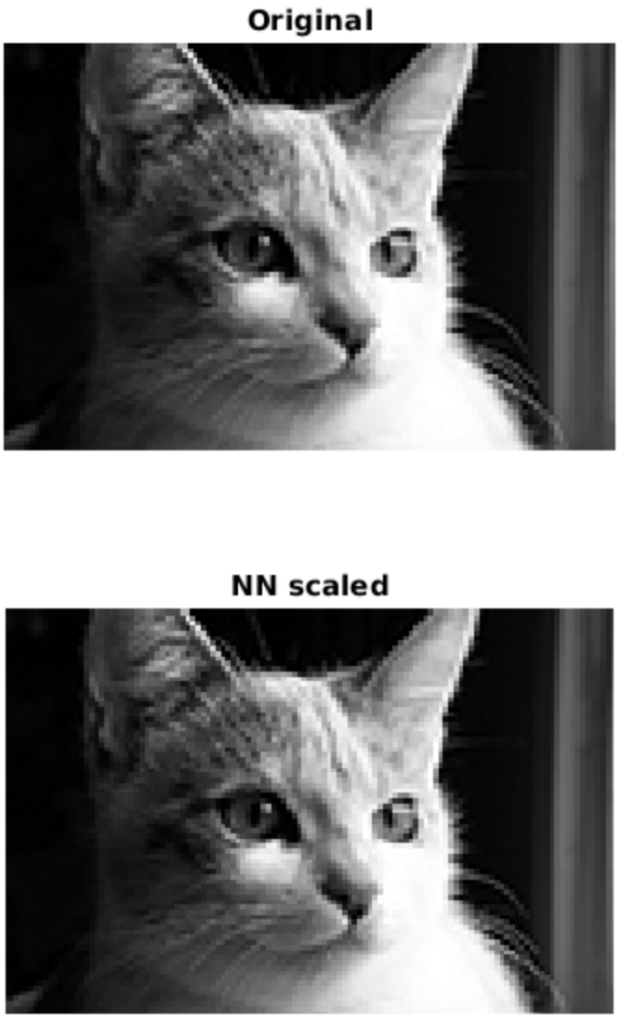

Fig. 2. Image scaled 13.11 times

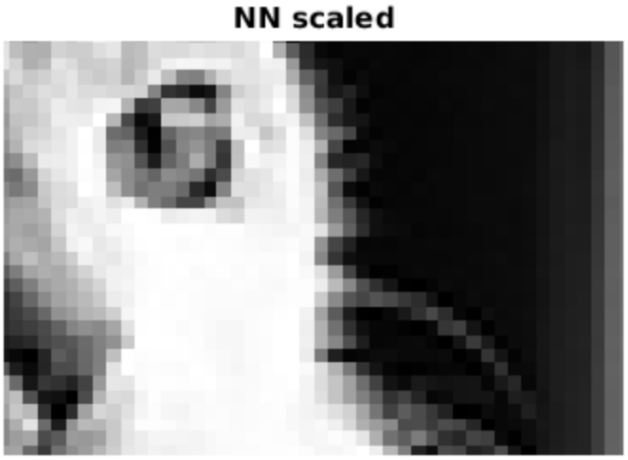

Fig. 3. Closeup on image scaled 13.11 times

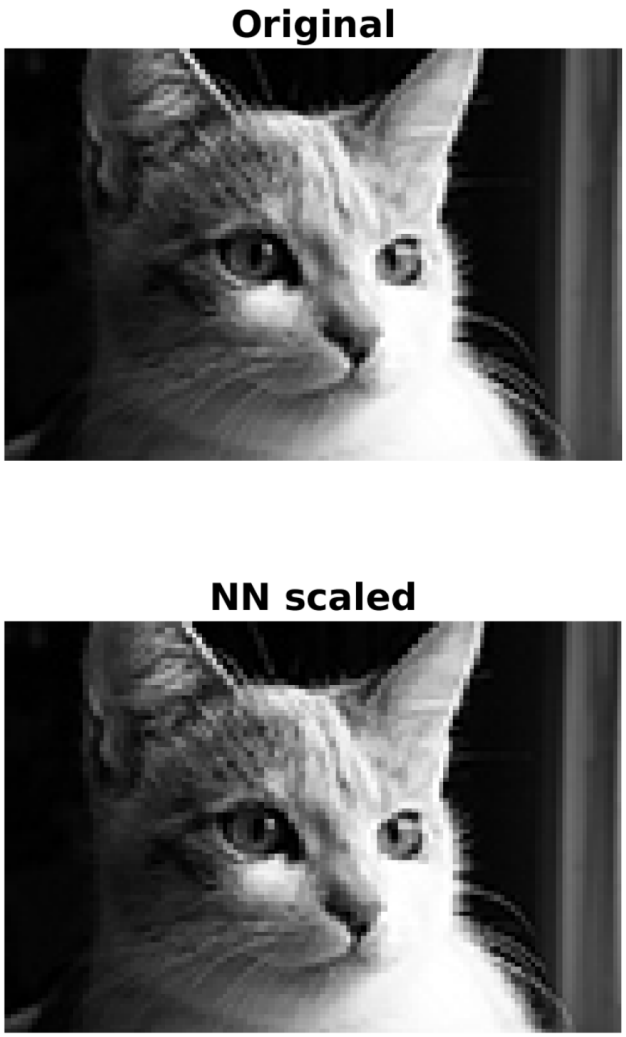

Fig. 4. Image scaled 2.5 times

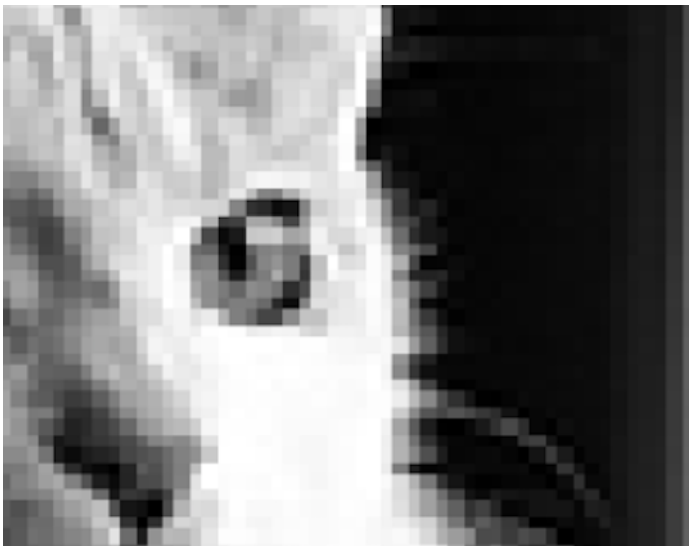

Fig. 5. Closeup on image scaled 2.5 times

C. bilinear interpolation

Unfortunately, I was not able to obtain satisfactory results for bilinear and bicubic interpolations. I tried to follow bilinear interpolation formulae given on Wikipedia, however the code never compiled due to exceeding array every time. I should have started solving this problem the first, not the last, so I could have contacted professor about the problems I faced before the deadline day.

II. Problem 2

Imported image was converted to black and white using function im2bw(). I tried to do this manually, but for some reason MATLAB kept going into infinite loop. As long as neighbors had to be accessed, there were errors about exceeding array. Those 8 exceptions for cases on image borderds to avoid exceeding array were created.

Boundary points are determined and given value of 1. Then a while() loop outside of original two loops was created, in order to iterate distance value dv. The full solution of the problem is as follows:

```
clear
dv=1; %counter for distance transform

bw = imread('horse.jpg');
[rows, cols] = size(bw);

bw=im2bw(bw);
%bw=(~bw); %in case the original image is white foreground on black background

[rows, cols] = size(bw);
dt = zeros(rows,cols); %blank matrix for distance transform image
dd=max(rows,cols); %maximum number of loops needed to perform distance transform
while dd>=dv
    for x = 1:rows
        for y = 1:cols
            if x==1&y==1  %% 8 exceptions for cases on image borderds to avoid
                exceeding array
                if (bw(x,y)==0) & ((bw(x+1,y)==1)|(bw(x,y+1)==1))
                    dt(x,y)=1;
                    continue;
                end
            end
            if x==1&y~=cols
                if (bw(x,y)==0) & ((bw(x+1,y)==1)|(bw(x,y+1)==1)|(bw(x,y-1)==1))
                    dt(x,y)=1;
                    continue;
                end
            end
            if x==1&y==cols
                if (bw(x,y)==0) & ((bw(x+1,y)==1)|(bw(x,y-1)==1))
                    dt(x,y)=1;
                    continue;
                end
            end
            if y==1&x~=rows
                if (bw(x,y)==0) & ((bw(x+1,y)==1)|(bw(x,y+1)==1)|(bw(x-1,y)==1))
                    dt(x,y)=1;
                    continue;
                end
            end
            if y==1&x==rows
                if (bw(x,y)==0) & ((bw(x+1,y)==1)|(bw(x-1,y)==1))
```

```
                    dt(x,y)=1;
                    continue;
                end
            end
            if y==cols&x~=1
                if (bw(x,y)==0) & ((bw(x+1,y)==1)|(bw(x-1,y)==1)|(bw(x,y-1)==1))
                    dt(x,y)=1;
                    continue
                end
            end
            if x==rows&y~=1
                if (bw(x,y)==0) & ((bw(x,y+1)==1)|(bw(x,y-1)==1)|(bw(x-1,y)==1))
                    dt(x,y)=1;
                    continue
                end
            end
            if y==cols & x==rows
                if (bw(x,y)==0) & ((bw(x,y-1)==1)|(bw(x-1,y)==1))
                    dt(x,y)=1;
                    continue
                end
            end
            if (bw(x,y)==0) && ((bw(x+1,y)==1)|(bw(x,y+1)==1)|(bw(x,y-1)==1)|(bw(x-1,
                y)==1))
                dt(x,y)=1; %assigning boundary points value of 1
            end
            if dv==1
                continue
            elseif (bw(x,y)==0) && dt(x,y)==0 && ((dt(x+1,y)==(dv-1))|(dt(x,y+1)==(dv
                -1))|(dt(x,y-1)==(dv-1))|(dt(x-1,y)==(dv-1)))
                dt(x,y)=dv; %assigning inner points value of dv, which iterates as we
                    go further from boundary points
            end
        end
    end
    dv=dv+1;
end

dmax=max(dt,[],'all');
for x = 1:rows
    for y = 1:cols
        dt(x,y)=round(dt(x,y)*(255/dmax)); %normalization
    end
end

check = bwdist(bw,'chessboard'); %check and compare
figure (1)
subplot(3,1,1), imshow(bw, []), title('Original')
hold on
subplot(3,1,2), imshow(dt, []), title('my algorithm')
hold on
subplot(3,1,3), imshow(check, []), title('bwdist "  hessboard  "')
hfig = figure (1)
print(hfig, '-dpng', '-r300', 'results')
```

After this, values were normalized between 0 and 255.

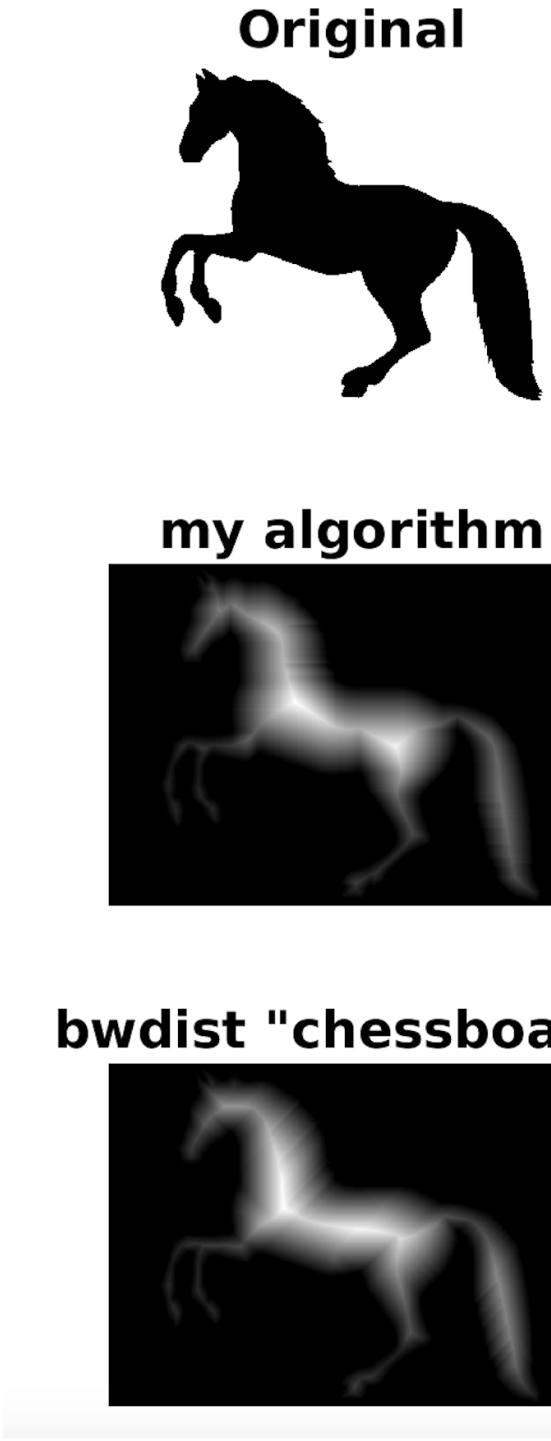

Fig. 6. horse image example

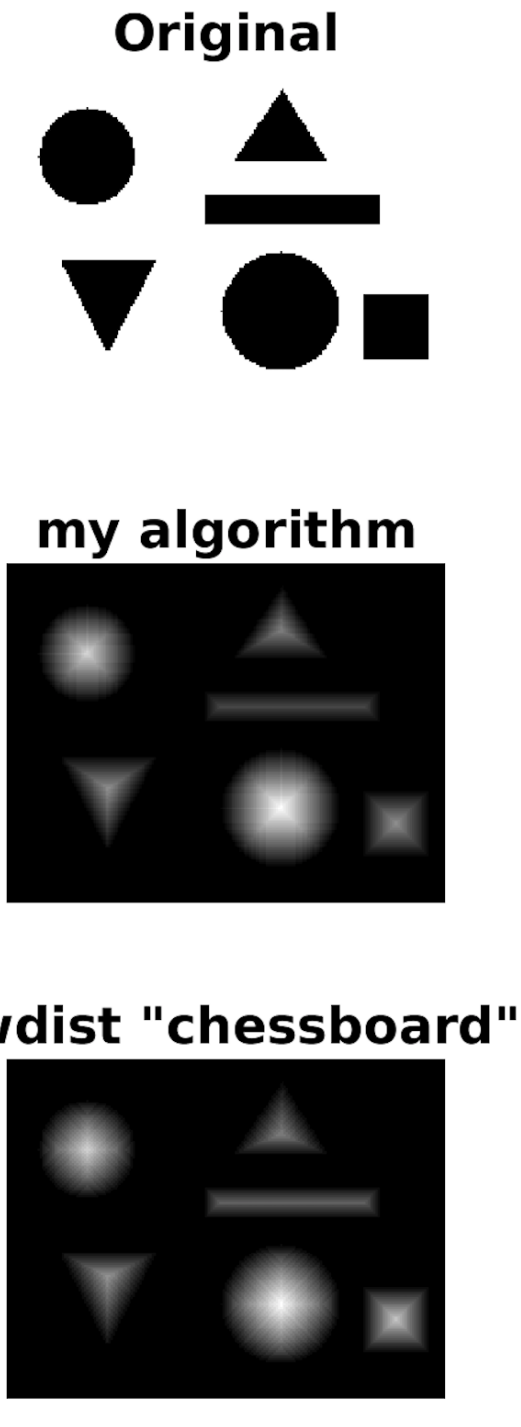

Fig. 7. coins image example

A. Results

I used bwdist(bw,'chessboard') function in order to check correctness of my results. The results have shown high similarity. The lines in distance transform had different directions in the examples, probably due to order of image traversal. Ultimately, the result was not affected.

III. Problem 3

Main part of the code from problem 2 was used in this solution in order to have matrix dt, which was an input to this function.

This problem seemed relatively easy at first glance. The corner pixels were easily determined as was described in the task. I used functions nonzeros() and unique() in order to sort the values in the matrix and obtain boundary values. Then, however, a problem arose with further neighbors. While it was easy to determine the value of 8 pixels around and compare them, I could not figure out how to track their coordinates at the same time. Thus, 8 "if" statements were created in order to be able to track the locations. Probably, there is more elegant solution possible, however I was not able to find it.

Further, my solution was only constructing skeleton for the upper half of the image. This was due to logic which compared nearest neighbors and selected the maximum (or the first one, if there was a tie). A solution to this problem was to have one more loop, this time inverted, in order to find skeleton for the lower part of the image. At last, I wanted to compare result of my algorithm with the bwskel() function, but unfortunately it was not working in my MATLAB version.

```
dtpos=[1,1];
s = zeros(rows,cols);
allvals = unique(nonzeros(dt));
boundary = min(unique(nonzeros(dt)));
for x = 1:rows
    for y = 1:cols
        if dt(x,y) == boundary && ((dt(x-1,y)==0)&&(dt(x,y-1)==0))|((dt(x-1,y)==0)&&(
            dt(x,y+1)==0))|((dt(x+1,y)==0)&&(dt(x,y-1)==0))|((dt(x+1,y)==0)&&(dt(x,y
            +1)==0))
             s(x,y)=dt(x,y);
        end
        if ((dt(x,y)~=0) && (s(x,y)~=0))
            neighbors=[(dt(x+1,y)),(dt(x,y+1)),(dt(x,y-1)),(dt(x-1,y)),(dt(x-1,y-1)),(
                dt(x-1,y+1)),(dt(x+1,y+1)),(dt(x+1,y-1))];
            [val,pos] = max(neighbors);
            if pos==1
              dtpos=[x+1,y];
            end
            if pos==2
              dtpos=[x,y+1];
            end
            if pos==3
              dtpos=[x,y-1];
            end
            if pos==4
              dtpos=[x-1,y];
            end
            if pos==5
              dtpos=[x-1,y-1];
            end
            if pos==6
              dtpos=[x-1,y+1];
            end
            if pos==7
              dtpos=[x+1,y+1];
            end
            if pos==8
              dtpos=[x+1,y-1];
            end
            s(dtpos(1),dtpos(2))=val;
```

```
        end
    end
end

for x = rows:-1:1
    for y = cols:-1:1
        if dt(x,y) == boundary && ((dt(x-1,y)==0)&&(dt(x,y-1)==0))|((dt(x-1,y)==0)&&(
            dt(x,y+1)==0))|((dt(x+1,y)==0)&&(dt(x,y-1)==0))|((dt(x+1,y)==0)&&(dt(x,y
            +1)==0))
             s(x,y)=dt(x,y);
        end
        if ((dt(x,y)~=0) && (s(x,y)~=0))
            neighbors=[(dt(x+1,y)),(dt(x,y+1)),(dt(x,y-1)),(dt(x-1,y)),(dt(x-1,y-1)),(
                dt(x-1,y+1)),(dt(x+1,y+1)),(dt(x+1,y-1))];
            [val,pos] = max(neighbors);
            if pos==1
              dtpos=[x+1,y];
            end
            if pos==2
              dtpos=[x,y+1];
            end
            if pos==3
              dtpos=[x,y-1];
            end
            if pos==4
              dtpos=[x-1,y];
            end
            if pos==5
              dtpos=[x-1,y-1];
            end
            if pos==6
              dtpos=[x-1,y+1];
            end
            if pos==7
              dtpos=[x+1,y+1];
            end
            if pos==8
              dtpos=[x+1,y-1];
            end
            s(dtpos(1),dtpos(2))=val;
        end
    end
end

for x = 1:rows
    for y = 1:cols
        s(x,y)=(s(x,y)/s(x,y)); %convert to bw
    end
end
figure(1)
imshow(s, [])
```

A. results

The solution showed good results in square and quadrilateral figures. Artifacts were noticed on all other figures, probably due to uneven corners. This proves that algorithm for corner determination has to be more sophisticated.

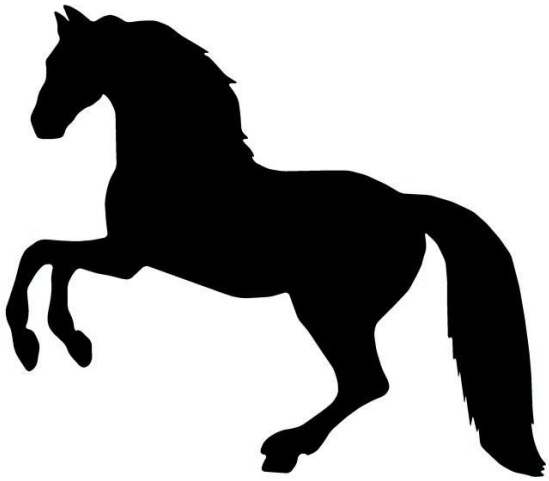

Fig. 8. original horse image example

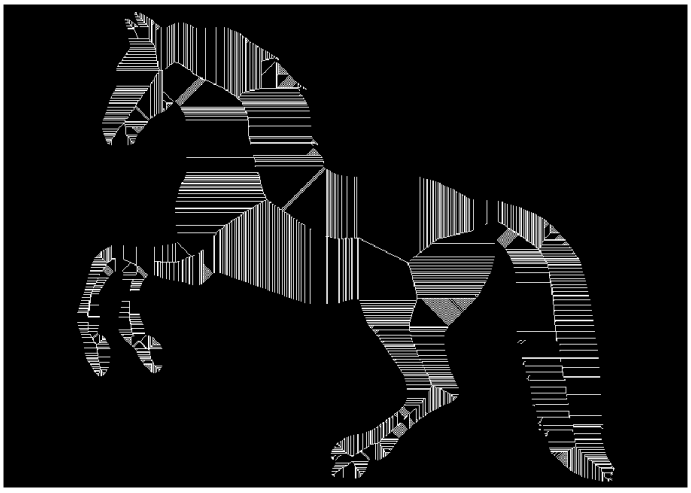

Fig. 9. horse image example

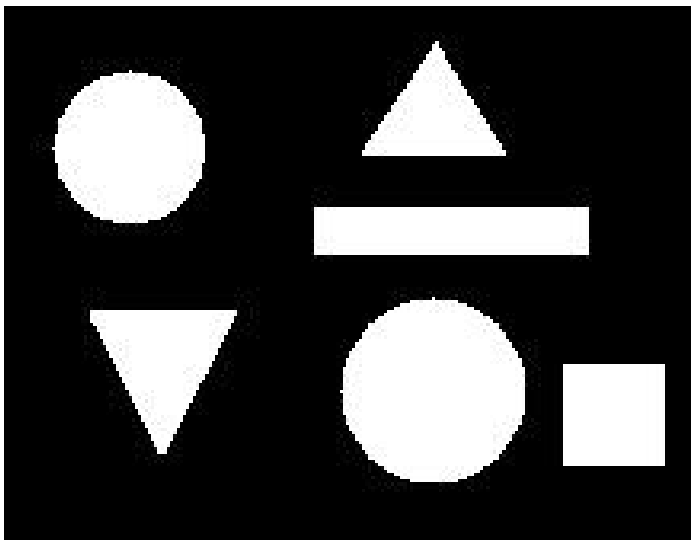

Fig. 10. original figures image example

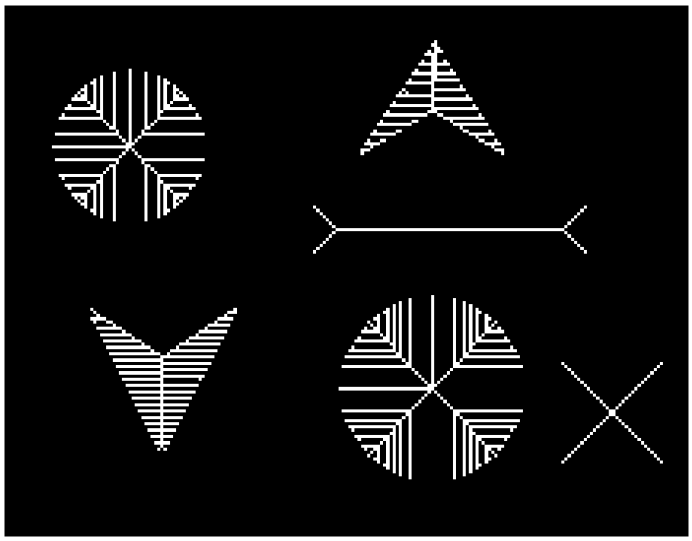

Fig. 11. figures image example

Overall, about 50 hours were spent solving these problems during 9 days. I believe that gained knowledge about MATLAB programming is mostly beneficial. However, I believe that more time for this amount of homework is needed, due to different backgrounds of students.