Image analysis - Lab 1

Adam Breitholtz

January 31, 2019

1.1

```
function patch=get_patch(image,x,y,patch_radius)

x0=x-patch_radius;
x1=x+patch_radius;
y0=y-patch_radius;
y1=y+patch_radius;
max_x=size(image,2);
max_y=size(image,1);
if(x0<=0||y0<=0||x1>max_x||y1>max_y)
    error("Patch not contained within image borders!");
end
z=size(image,3); % deal with multilayer images
patch=zeros(y1-y0+1,x1-x0+1,z);
for k=1:z
   patch(:,:,k)=image(y0:y1,x0:x1,k);
end
end
```

1.2

```
% patch function

test_image=reshape((11:100),10,9);
test_image(5,5)=0;

patch=get_patch(test_image,5,5,3)
imagesc(patch)
```

1.3

See 1.1

1.4

```
function output=gaussian_filter(greyscale,std)
% size of filter given
```

```
size=4*std;
filter=fspecial('gaussian',[size size],std);
output=imfilter(greyscale,filter,'symmetric');
end
```

1.5

```
function [grad_x, grad_y]=gaussian_gradients(img,std)
filtered=gaussian_filter(img,std);
% then take the 'laplacian of the image'
[grad_x, grad_y]=gradient(filtered);
end
```

1.7

```
function histogram=gradient_histogram(grad_x,grad_y)
bin=zeros(8,1);
for i=1:size(grad_x,1)
    for j=1:size(grad_x,2)
dir=atan2(grad_y(i,j),grad_x(i,j)); % direction in radians from the positive x-axis
len=sqrt(grad_x(i,j)^2+grad_y(i,j)^2);
if dir<=pi/4 && dir>0
    bin(1)=bin(1)+len;
else if dir<=2*pi/4 && dir>pi/4
        bin(2)=bin(2)+len;
    else if dir<=3*pi/4 && dir>2*pi/4
            bin(3)=bin(3)+len;
        else if dir<=pi && dir>3*pi/4
                bin(4)=bin(4)+len;
                % here we change it up a bit since arctan takes values in
                % [-pi,pi]; so now we go from -pi to 0 to keep consistent
            else if dir<=-3*pi/4 && dir>-pi
                    bin(5)=bin(5)+len;
                else if dir<=-2*pi/4 && dir>-3*pi/4
                        bin(6)=bin(6)+len;
                    else if dir<=-pi/4 && dir>-2*pi/4
                            bin(7)=bin(7)+len;
                        else
                            bin(8)=bin(8)+len;
                        end
                    end
                end
            end
        end
    end
end
% end of stupid if case
    end
end
histogram=bin;
end
```

1.8

```
function region_centres = place_regions(centre,radius)
% given [y;x] centre of a region of interest
% create 3*3 regions with radius=radius around the centre
% output the centre points of all the
y=centre(1);
x=centre(2);
D=2*radius+1;

region_centres=[y-D y y+D y-D y y+D y-D y y+D;
    x-D x-D x-D x x x x+D x+D x+D];
end
```

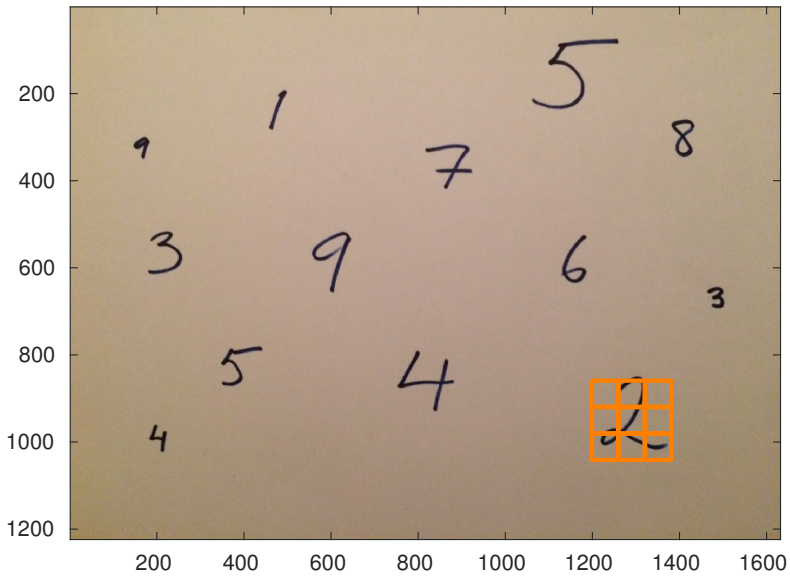

Figure 1: One output of the plot_squares function.

1.9

```
function desc=gradient_descriptor(image,position,radius)
histogram=zeros(72,1);
% divide up space around position into regions
centres=place_regions(position,radius);
% loop over regions
for i=1:8
patch=get_patch(image,centres(1,i),centres(2,i),radius);

% compute gaussian gradients with std prop. to radius
```

```
% std=(1/10)*radius is chosen to get a good approx after empirical tests
[grad_x,grad_y]=gaussian_gradients(patch,ceil(radius./10));

% compute histograms for the region and put into vector
histogram(i*9-8:(i*9)-1)=gradient_histogram(grad_x,grad_y);


end

% normalise vector to unit length
desc=normalize(histogram);

end
```

1.10

```
%% prepare digits
load('digits.mat');

% 39*39 images -> radius 6 & position (20,20)
desc_train=zeros(72,100);

% compute descriptors
for i=1:100
    desc_train(:,i)=gradient_descriptor(digits_training(i).image,[20;20],6);
end

% save into struct
N=mat2cell(desc_train,72,ones(1,100));
[digits_training(:).desc]=N{:};
```

1.11

```
function label=classify_digit(digit_image,digits_training)
% go through saved descriptors and take norm difference
S=size(digits_training,2);
diff=zeros(S,1);
descriptor=gradient_descriptor(digit_image,[20;20],6);
prepare_digits;
for i=1:S
    % if you cannot be assumed to have used prepare_digits
    % then exchange desc2 with
    % desc2=gradient_descriptor(digits_training(i).image,[20;20],6);
   desc2=digits_training(i).desc;
   diff(i)=norm(desc2-descriptor);
end
[M,I]=min(diff);
label=digits_training(I).label;
end
```

1.12

This function outputs 37/50 or 74% correct classifications with the standard deviation being $\frac{radius}{10}$.

```
%% classify_all_digits
sum=0;
% take each image in validation and classify it
S=size(digits_validation,2);
for i=1:S
    label=classify_digit(digits_validation(i).image,digits_training);

    if(strcmp(num2str(label),num2str(digits_validation(i).label)))
        sum=sum+1;
    end
end
disp(['Amount correct: ' num2str(sum) ' out of ' num2str(S)])
percent=sum/S;
disp(['Percent correct:' num2str(100*percent) '%'])
```

1.13

Yes, it works for all four if you tune the radius parameter such that it fits the number well. Radii that work are 30 for the first three and 8 for the last.

How would we automate the setting of the radius parameter? We perhaps could use something like the Difference of Gaussians method to identify the numbers as blobs in the image and find the radius from that information.

1.15

```
function [label,name]=classify_church(image,feature_collection)
[coords1, desc1]=extractSIFT(image);
corrs=matchFeatures(desc1',feature_collection.descriptors','MatchThreshold',100,'MaxRatio',0.7);
K=size(corrs,1);
L=zeros(K,1);
for i=1:K
    L(i)=feature_collection.labels(corrs(i,2));
end
% find most occurring label
HIST=hist(L,5);
[M,I]=max(HIST);
label=I;
name=feature_collection.names{I};
end
```

1.16

Classification of the church images yielded the correct answer for all 10 churches. Example code for the computation for one church is below.

```
im1=imread('church10.jpg');
[LABEL,NAME]=classify_church(double(rgb2gray(im1)),feature_collection)
manual_labels{10}
```