Yongjin Shin(IME, 20090488)

1. Vanishing Point Estimation in Image

1) 목적

본 과제의 목적은 Vanishing Point 를 찾아내는 것이다. Vanishing Point 는 n 개의 직선의 교점에서 찾을 수 있는데, 이 과제에서는 가장자리(Edge)를 detection 한 뒤에 가장자리의 교점들 중 가장 많은 빈도로 나타나는 교점을 Vanishing Point 로 확인할 수 있다.

Image1 과 Image2 의 코드는 parameter 의 값들을 제외하면 동일하기 때문에 image 1 의 코드를 통해 결과를 해석해보도록 하겠다.

2) 결과

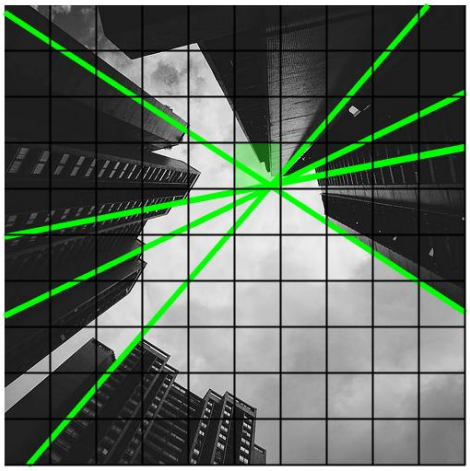

image1_output.jpg

image2_output.jpg

3) 코드 분석

```
%load image
img = imread('image1.jpg');
img = rgb2gray(img);

%performing canny dection and hough transform
BW = edge(img, 'canny');
[H,theta,rho] = hough(BW,'RhoResolution', 0.5 ,'Theta', -89:89);
```

우선, 이미지를 불러온 뒤에 RGB channel 을 Gray Channel 로 변환시킨다. 그리고 Canny Edge detection 을 통해 Edge 을 찾아낼 수 있다. Edge 들 중에서 Line 을 이루는 점들을 걸러내기 위해서 Hough transform 을 시행하는데, 이 때 origin (1,1) 에서부터 Line 까지의 거리를 Rho Resolution parameter 를 통해 설정하게 된다. Theta 는 origin 에서 Line 까지의 수선이 이루는 각을 뜻하는데, image1 에서는 모든 방향에서 line 이 나타나야 하므로 한계 값이 없도록 설정하였다.

```
%find out hough peak and lines
P = houghpeaks(H,4,'threshold',ceil(0.1*max(H(:))));
lines = houghlines(BW,theta,rho,P,'FillGap',5);
```

houghpeaks 함수를 통해서 hough function 의 max 값 중 일부를 찾아낸다. 여기서는 max 값의 0.1 만큼 threshold 를 주고 이 중에서 4 개를 선택하도록 하였다. 여기서 얻어진 값들을 통해 line 을 이루는 점들을 찾을 수 있다. P 에서 선택된 값들과 hough 함수를 통해 얻은 theta 와 rho 값을 parameter 로 하여 line 을 규명한다. 결과값은 structure 로 저장된다.

```
%draw lines
figure, imshow(img), hold on
li = zeros(length(lines), 2);
x = get(gca,'XLim');
for k = 1:length(lines)
    %line
    x1  = [lines(k).point1(1) lines(k).point2(1)];
    y1  = [lines(k).point1(2) lines(k).point2(2)];

    %fit linear polynomial
    p1 = polyfit(x1,y1,1);
    li(k,:) = p1;
    plot(x, p1(1)*x+p1(2), 'LineWidth', 3, 'Color', 'green');
end
```

Line 을 이루는 두 점들이 li 의 하나의 structure 로 구성되어 있다. 우리는 이 두 점들의 x, y 값을 통해서 실제 line 을 그려낼 수 있다.

```
%calculate intersection
intersect = [];
for k = 1:length(li)
    for l=1:length(li)
        if l ~= k
            x_intersect = fzero(@(x) polyval(li(k,:)-li(l,:),x),3);
            y_intersect = polyval(li(k,:), x_intersect);
            intersect = [intersect;[x_intersect y_intersect]];
        end
    end
end
```

한 line 이 자신을 제외한 다른 line 들과 접하는 교점을 모두 구한다. 이를 통해 가장 많은 vote 를 받은 점이 vanishing point 가 된다.

```
%get the max value of most voted point
BinW=50;BinH=50;
[N,Xedges,Yedges] = histcounts2(intersect(:,1),intersect(:,2),...
    'BinWidth',[BinW BinH], 'XBinLimits', [0, 320], 'YBinLimits', [0,560]);
maxPoint = max(N(:));
[maxRow, maxCol] = find(N == maxPoint);
```

이미지를 일정 크기의 Bin Size 로 나눈 뒤에 교점들이 지닌 값을 통해 voting map 을 가지도록 한다. 이를 통해 어떤 bin 에서 가장 많은 voting 이 이루어졌는지 알 수 있다.

2. Object Classification

1) 목적

본 과제를 통해서 Support Vector Machine 을 통해 Class 을 나누는 작업을 진행한다. 이 때, 사용되는 feature 는 Histogram of Oriented Gradients 을 이용하여 이미지들의 feature 을 추출하도록 한다. 여기서 우리는 5 가지의 이미지를 사용하기 때문에, Label 은 5 개가 된다. 따라서 One vs Rest 전략을 통해 Classify 을 진행한다.

Dataset 은 각 Label 별로 50 개의 이미지를 사용하고 이 중 각 30 개는 Training Set, 나머지 20 개씩의 이미지는 Test Set 로 사용이 된다.

Label 1 = airplane
Label 2 = Car side
Label 3 = Chair
Label 4 = Elephant
Label 5 = Ferry

2) 결과

airplane : Accuracy = 100% (100/100) (classification)
Car side : Accuracy = 100% (100/100) (classification)
Chair : Accuracy = 85% (85/100) (classification)
Elephant : Accuracy = 95% (95/100) (classification)
Ferry :Accuracy = 86% (86/100) (classification)

acc = 0.8600

C =

20	0	0	0	0
0	20	0	0	0
1	0	12	3	4
0	0	0	19	1
0	0	0	5	15

위의 Confusion Table 에 의해 Airplane 과 Car Side 는 20 개의 test set 모두 classification 이 성공했음을 확인할 수 있다 (100%). Chair 의 경우, 12 개만 정확하게 맞추었고 Airplane/ Elephant/ Ferry 로 혼동을 하는 것을 확인가능하다. 나머지도 비슷하게 해석이 가능하다.

3) 코드 분석

```
final_dir = 'dataset/';

finalSet = imageDatastore(final_dir, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
finalSet.ReadFcn = @(trainingSet)imresize(imread(trainingSet),[256 256]);
[trainingSet,testSet] = splitEachLabel(finalSet,0.6);
trainingSet = shuffle(trainingSet);
testSet = shuffle(testSet);

cellSize = [5 5];
img = readimage(trainingSet, 87);
[hog_4x4] = hog_feature_vector(img);
hogFeatureSize = length(hog_4x4);

numLabels = 5;
numImages = numel(trainingSet.Files);
trainingFeatures = zeros(numImages, hogFeatureSize, 'double');
trainingLabels = grp2idx(trainingSet.Labels);
```

dataset 폴더 내에서 folder name 에 따라서 Label 을 나누었다. 이 dataset 중에서 앞의 60%는 training set 으로 사용되며, 40%는 test set 으로 사용되었다. 이렇게 나누어진 Training Set 과 Test Set 은 [256, 256] size 로 사이즈가 조절된 이후 순서가 무작위로 Shuffle 된 이후에 사용된다.

```
for i = 1:numImages
    img = readimage(trainingSet, i);
    trainingFeatures(i, :) = hog_feature_vector(img);
end

% train one vs all models
model = cell(numLabels, 1);
for k=1:numLabels
    model{k} = svmtrain(double(trainingLabels == k), trainingFeatures, '-b 1');
end
```

모든 Training Set 에 있는 이미지들은 HOG 를 통해 Feature 를 추출해 낸다. 이를 통해 추출한 150 개 이미지들의 feature 들을 통해 각 label 에 해당하는 모델을 train 할 수 있다. Train 은 one vs rest 전략을 취하는데, 위의 코드와 같이 해당 label 에 적합한 feature 는 1 로 잡고 그 이외의 feature 들은 0 으로 Labelling 이 이루어진다. 이를 각각의 label 에 적용한다. 예측 값을 알기 위해서 ProbA 와 ProB 를 알아야만 한다. 따라서, '-b 1' 옵션을 활용한다. 이는 svmpredictd 에서도 사용하여 probability estimate vector 를 추출한다.

```
angle=atand(Ix./Iy); % Matrix containing the angles of each edge gradient
angle=imadd(angle,90); %Angles in range (0,180)
magnitude=sqrt(Ix.^2 + Iy.^2);

% Remove redundant pixels in an image.
angle(isnan(angle))=0;
magnitude(isnan(magnitude))=0;

feature=[]; %initialized the feature vector
```

HOG 는 각 픽셀에서 방향과 크기를 가진 gradient vector 를 추출하여 feature 를 만들어낸다.

```
for i = 0: rows/B_WIDTH - 2
    for j= 0: cols/B_WIDTH -2

        mag_patch = magnitude(B_WIDTH*i+1 : B_WIDTH*i+10 , B_WIDTH*j+1 : B_WIDTH*j+10);
        %mag_patch = imfilter(mag_patch,gauss);
        ang_patch = angle(B_WIDTH*i+1 : B_WIDTH*i+10 , B_WIDTH*j+1 : B_WIDTH*j+10);

        block_feature=[];

        %Iterations for cells in a block
        for x= 0:1
            for y= 0:1
                angleA =ang_patch(B_WIDTH*x+1:B_WIDTH*x+B_WIDTH, B_WIDTH*y+1:B_WIDTH*y+B_WIDTH);
                magA   =mag_patch(B_WIDTH*x+1:B_WIDTH*x+B_WIDTH, B_WIDTH*y+1:B_WIDTH*y+B_WIDTH);
                histr  =zeros(1,9);

                %Iterations for pixels in one cell
                for p=1:B_WIDTH
                    for q=1:B_WIDTH
%
                        alpha= angleA(p,q);
```

Block size 가 [5 5]이므로 B_width 는 5 를 뜻하며, stide 가 50%이므로 각 픽셀들을 단위 5 씩 이동하면서 feature 들을 추출하게 된다. 이 때 bin size 가 9 이므로 20 씩 끊어져서 bin 을 구성하게 된다.

```
block_feature=block_feature/sqrt(norm(block_feature)^2+.001);
for z=1:length(block_feature)
    if block_feature(z)>0.2
         block_feature(z)=0.2;
    end
end
block_feature=block_feature/sqrt(norm(block_feature)^2+.001);
feature=[feature block_feature]; %Features concatenation
```

이렇게 하나의 block 에 feature 들을 추출한 뒤, L2 Normalization 을 시행하게 된다. 이를 통해 보다 빛에 덜 민감한 feature 를 만들어낼 수 있다. L2 Normalization 이후 feature 가 0.2 보다 큰 경우는 0.2 로 조정을 한 뒤 다시 Normalization 을 거친다. (L2 Hys)

```
% get prob. estimates of test instance using each model
numTest = numel(testSet.Files);
testLabels = grp2idx(testSet.Labels);
prob = zeros(numTest, numLabels);
p = zeros(numTest, 2);
testFeatures = zeros(numTest, hogFeatureSize, 'double');

for i = 1:numTest
    img = readimage(testSet, i);
    testFeatures(i, :) = hog_feature_vector(img);
end
```

Train Set 과 동일하게 Test Set 에서도 HoG Feature 를 추출해낸다.

```
for k=1:numLabels
    [~,~,p] = svmpredict(double(testLabels == k), testFeatures, model{k}, '-b 1');
    prob(:,k) = p(:,model{k}.Label == 1);
end

% predict the class with the highest probability
[~, pred] = max(prob, [], 2);
acc = sum(pred == testLabels) ./ numel(testLabels)
C = confusionmat(testLabels, pred)
```

추출한 Test Feature 를 모델을 통하여 각 Label 별로 예측을 한다.