Artificial Intelligence/Machine Learning - Coursework
Robert Parry - 19028639

1) Filesize - based features ~ Working
2) Brightness - based features ~ Working
3) Edge - based features ~ Not fully working
4) HOG1 - based features ~ Working
5) BoVW2 - based features ~ Not fully working - worked once then stopped
6) CNN - based features
  i) resnet50 ~ Working
  ii) alexnet ~ Working
7) Extra - KNN ~ Working
8) Functions

1. Filesize-based features

```
%Reading the file sizes of each picture, then inputting them into a
table
    % load the dataset:
    data = readcell('MerchData.csv');

    % Create the dataset array
    data(1:1:10, 1:1:end)
    data(1, :) = [];

    % Populate the array with the data
    data = data(randperm(size(data,1)), :);


    rng(0); % please leave this re-seeding of the random number
generator in place so we can compare results

    % Set the variable values using 'data'
    nTest = round(0.4 * size(data,1))
    data_test = data(1:1:nTest, :);
    data_train = data(nTest+1:1:end, :);


    %Set the label index based on how many label columns needed
    label_index = 2;
    %Set test_labels/examples data using the previous data test and
train vars
    test_labels = categorical(data_test(:, label_index));
    test_examples = cell2mat(data_test(:, 1:end~=label_index));
    %Set train_labels/examples data using the previous data test and
train vars
    train_labels = categorical(data_train(:, label_index));
    train_examples = cell2mat(data_train(:, 1:end~=label_index));


    %Set var 'm_knn'  to use the function for 'knn' using the two
stated variables
    m_knn = fitcknn(train_examples, train_labels)
```

```
    %Set var 'm_nb'  to use the function for 'cnb' using the two
stated variables
    m_nb = fitcnb(train_examples, train_labels)

    %Set var 'm_dt'  to use the function for 'tree' using the two
stated variables
    m_dt = fitctree(train_examples, train_labels)

    %Set var 'm_ann'  to use the function for 'net' using the two
stated variables
    m_ann = fitcnet(train_examples, train_labels)

    %Set var 'm_svm'  to use the function for 'ecoc' using the two
stated variables
    m_svm = fitcecoc(train_examples, train_labels)


%Prediction Setup
    %Using the trained classifier to predict using the
'test_examples'
    predictions = predict(m_knn, test_examples);

    %Using the trained classifier to predict using the
'test_examples'
    %Set var 'predictions1'  to use the function for predict using
the two stated variables
    predictions1 = predict(m_nb, test_examples);

    %Using the trained classifier to predict using the
'test_examples'
    %Set var 'predictions2'  to use the function for predict using
the two stated variables
    predictions2 = predict(m_dt, test_examples);

    %Using the trained classifier to predict using the
'test_examples'
    %Set var 'predictions3'  to use the function for predict using
the two stated variables
    predictions3 = predict(m_ann, test_examples);

    %Using the trained classifier to predict using the
'test_examples'
    %Set var 'predictions4'  to use the function for predict using
the two stated variables
    predictions4 = predict(m_svm, test_examples);


%KNN Evaluation

    %Create a confusion matrix using 'test_labels' and 'predictions'
    [c, order] = confusionmat(test_labels, predictions)

    %Create and display a confusion chart using 'test labels' and
predictions
    confusionchart(test_labels, predictions)
```

```
    %Output accuracy
    p = sum(diag(c)) / sum(c(1:1:end))

%NB Evaluation

    [c, order] = confusionmat(test_labels, predictions1)

    confusionchart(test_labels, predictions1)

    p = sum(diag(c)) / sum(c(1:1:end))

%DT Evaluation

    [c, order] = confusionmat(test_labels, predictions2)

    confusionchart(test_labels, predictions2)

    p = sum(diag(c)) / sum(c(1:1:end))

%ANN Evaluation

    [c, order] = confusionmat(test_labels, predictions3)

    confusionchart(test_labels, predictions3)

    p = sum(diag(c)) / sum(c(1:1:end))

%SVM Evaluation
    [c, order] = confusionmat(test_labels, predictions4)

    confusionchart(test_labels, predictions4)

    p = sum(diag(c)) / sum(c(1:1:end))
```

2. Brightness-based features

```
%Data
    %Check the dataset is present:
    if ~exist('./MerchData', 'dir')
        error('MerchData does not exist');
    end
```

```
%Working with the datastore
    %Create the imagedatastore object:
    imds = imageDatastore('MerchData', 'IncludeSubfolders', true);

    % loop over all the images extracting their size
    grow = [];
    imds.reset();
    while hasdata(imds)
        im = imds.read();
        grow(end+1,:) = size(im); % store the dimensions of the
image (height, width, depth)
    end

    size(grow)



%Splitting image dataset
    rng(0);

    %Using 'MerchData' to populate 'imds'
    imds = imageDatastore('MerchData', 'IncludeSubfolders', true,
'LabelSource','foldernames')
    [training, testing] = splitEachLabel(imds, 0.6, 'randomize');


    %Set 'im' to the first image from training
    im = training.readimage(1)
    %im = my_im2gray(im)
    imshow(im);

    %Use the size function with 'im' as the input
    size(im)


    %Convert the loaded image from rgb(colour) to
greyscale(black&white)
    im = im2gray(im);
    size(im)
    x = size(im)
    n = x(1,2) * x(1,2)
    %Output the loaded image
    imshow(im);


%Testing code
    %Read the image and put it in an array
    my2dimagearray = imread('cameraman.tif');
    imshow(my2dimagearray);


    %Changing the brightness value of the 100th value
    my2dimagearray(100,50) = 255;
    imshow(my2dimagearray)
```

```
% %Working with the image data - testing
%
%     %Load the first image from the dataset into the var 'im'
%     im = training.readimage(1)
%     imshow(im);
%
%     %calculate average brightness of an image
%     brightness = mean2(im)
%
%     %Create and populate the array 'imageArray' with the data from
'im'
%     imageArray = []
%     imageArray = im
%
%     %Create and populate the var 'summedImage' with the summed
values from 'ImageArray'
%     summedImage = sum(imageArray(),"all")
%
%     finbri = summedImage / n
```

```
%Setting up and running the training data
    %Populate the training data using the images in the dataset and
extracting
    %the relavent data using the 'get_brightness()' function
    train_examples = [];
    while hasdata(training)
    %Convert the training image to greyscale
    im = im2gray(training.read());
    %Populate 'train_examples' using the 'get_brightness' function
    train_examples(end+1,:) = get_brightness(im);
    end


    train_labels = training.Labels;
```

```
%Setting up and running the testing data
    %Populate the test examples using the images in the dataset and
extracting
    %the relavent data using the 'get_brightness()' function
    test_examples = [];
    while hasdata(testing)
    %Set 'im' to a greyscale version of the training image loaded
    im = im2gray(testing.read());
    %populate the 'test_examples' with the image data that has been
run
    %through the 'get_brightness()' function
    test_examples(end+1,:) = get_brightness(im);
```

```
    end


    %Copy the testing label data over
    test_labels = testing.Labels;

    %Check the data is present and correct by outputting it
    test_examples


%Setting up Classifier and Prediction
    %Set var 'm_knn'  to use the function for 'knn' using the two
stated variables
    m_knn = fitcknn(train_examples, train_labels , 'NumNeighbors',
3)

    %Set var 'predictions'  to use the function for predict using
the two stated variables
    predictions = predict(m_knn, test_examples);




%Evaluate classifiers performance


    %Create a confusion matrix using 'test_labels' and 'predictions'
    [c, order] = confusionmat(test_labels, predictions)

    %Create and display a confusion chart using 'test labels' and
predictions
    confusionchart(test_labels, predictions)

    %Output accuracy
    p = sum(diag(c)) / sum(c(1:1:end))
```

3. Edge-based features

```
%Check the dataset is present:
    if ~exist('./MerchData', 'dir')
        error('MerchData does not exist');
    end


rng(0);


%Working with the datastore
    %Create the imagedatastore object:
```

```
    imds = imageDatastore('MerchData', 'IncludeSubfolders', true);

    % loop over all the images extracting their size
    grow = [];
    imds.reset();
    while hasdata(imds)
        im = imds.read();
        grow(end+1,:) = size(im); % store the dimensions of the
image (height, width, depth)
    end

    % here's the data we're left with...
    size(grow)

    %set 'im' to image 1
    im = training.readimage(1);
    imshow(im);

    %Convert image to greyscale
    im = im2gray(im);
    imshow(im);

%Setting up and running the training data
    train_examples = [];
    %Loop until all of the training data has been used
    while hasdata(training)
    %Convert the training image to greyscale
    im = im2gray(training.read());
    %Populate train_exaples using the 'get_edges' function
    train_examples(end+1,:) = get_edges(im);
    end


    %Set 'train_labels' equal to 'training.labels'
    train_labels = training.Labels;



%Setting up and running the testing data
    test_examples = [];
    while hasdata(testing)
    %Set 'im' to a greyscale version of the training image loaded
    im = im2gray(testing.read());
    %Populate 'test_examples' using the 'get_edges' function
    test_examples(end+1,:) = get_edges(im);
    end


    %Set the 'test_labels' equal to 'training.labels'
    test_labels = testing.Labels;



%Setting up Classifier and Prediction
```

```
    %Set var 'm_knn'  to use the function for 'knn' using the two
stated variables
    m_knn = fitcknn(train_examples, train_labels, 'NumNeighbors', 3)

    %Using the trained classifier to predict using the
'test_examples'
    predictions = predict(m_knn, test_examples);
```

```
%Evaluate classifiers performance
```

```
    %Create a confusion matrix using 'test_labels' and 'predictions'
    [c, order] = confusionmat(test_labels, predictions)

    %Create and display a confusion chart using 'test labels' and
predictions
    confusionchart(test_labels, predictions)

    %Output accuracy
    p = sum(diag(c)) / sum(c(1:1:end))
```

4. HOG-based features

```
%Check the dataset is present:
    if ~exist('./MerchData', 'dir')
        error('MerchData does not exist');
    end
```

```
rng(0);%Reset the random number generator
```

```
%Working with the datastore
    %Create the imagedatastore object:
    imds = imageDatastore('MerchData', 'IncludeSubfolders', true);
```

```
    % loop over all the images extracting their size
    grow = [];
    imds.reset();
    while hasdata(imds)
        im = imds.read();
        grow(end+1,:) = size(im); % store the dimensions of the
image (height, width, depth)
    end
```

```
    size(grow)


%Splitting image dataset
    imds = imageDatastore('MerchData', 'IncludeSubfolders', true,
'LabelSource','foldernames')
    [training, testing] = splitEachLabel(imds, 0.6, 'randomize');
    % <my comment removed>:
    im = training.readimage(1);
    imshow(im);



% write your code on the lines below:
im = im2gray(im);
imshow(im);


[Gx, Gy] = imgradientxy(im, 'Prewitt');
[Gmag, Gdir] = imgradient(Gx, Gy);


imagesc(Gmag);
imagesc(Gdir);


CELLSIZE = [16 16];
[h, v] = extractHOGFeatures(im, 'CellSize', CELLSIZE, ...
'BlockSize', [floor(size(im,1)/CELLSIZE(1)) floor(size(im,2)/
CELLSIZE(2))], ...
'UseSignedOrientation', true);



imshow(im);
hold('on');
v.plot();



%Setting up and running the training data
    train_examples = [];
    %Loop until all of the training data has been used
    while hasdata(training)
    %Set 'im' to a greyscale version of the training image loaded
    im = im2gray(training.read());
    %Populate 'train_examples' using the 'get_hogs' function
    train_examples(end+1,:) = get_hogs(im);
    end
```

```
    train_labels = training.Labels;



%Setting up and running the testing data
    test_examples = [];
    %Loop until all of the testing data has been used
    while hasdata(testing)
    %Set 'im' to a greyscale version of the image loaded
    im = im2gray(testing.read());
    %Populate 'test_examples' using the 'get_hogs' function
    test_examples(end+1,:) = get_hogs(im);
    end


    test_labels = testing.Labels;


%Setting up Classifier and Prediction
    %Set var 'm_knn'  to use the function for 'knn' using the two
stated variables
    m_knn = fitcknn(train_examples, train_labels , 'NumNeighbors',
3)

    %Using the trained classifier to predict using the
'test_examples'
    predictions = predict(m_knn, test_examples);




%Evaluate classifiers performance
    %Create a confusion matrix using 'test_labels' and 'predictions'
    [c, order] = confusionmat(test_labels, predictions)

    %Create and display a confusion chart using 'test labels' and
predictions
    confusionchart(test_labels, predictions)

    %Output accuracy
    p = sum(diag(c)) / sum(c(1:1:end))
```

5. BoVW-based features

```
%Check the dataset is present:
    if ~exist('./MerchData', 'dir')
        error('MerchData does not exist');
    end
```

```
%Reset the random number generator to 0
rng(0);


%Working with the datastore
    %Create the imagedatastore object:
    imds = imageDatastore('MerchData', 'IncludeSubfolders', true);

    % loop over all the images extracting their size
    grow = [];
    imds.reset();
    while hasdata(imds)
        im = imds.read();
        grow(end+1,:) = size(im); % store the dimensions of the
image (height, width, depth)
    end



    size(grow)


    %set 'im' to the first training image
    im = training.readimage(1);
    imshow(im);

    im = im2gray(im);

    H = get_hogs(im);

    %Testing words.encode
    train_examples(end+1,:) = words.encode(im);
    train_examples


%Setting up and running the training data
    train_examples = [];
    %Loop until all of the training data has been used
    while hasdata(training)
    %Set 'im' to a greyscale version of the training image loaded
    im = im2gray(training.read());
    % you can call the .encode() method directly (rather than your
own function):
    train_examples(end+1,:) = words.encode(im);
    end


    % <my comment removed>:
    train_labels = training.Labels;



%Setting up and running the testing data
    test_examples = [];
    %Loop until all of the testing data has been used
```

```
    while hasdata(testing)
    % <my comment removed>:
    im = im2gray(testing.read());
    % you can call the .encode() method
    % of the object you just created:
    test_examples(end+1,:) = words.encode(im);
    end


    % <my comment removed>:
    test_labels = testing.Labels;


%Setting up Classifier and Prediction
    %Set var 'm_knn'  to use the function for 'knn' using the two
stated variables
    m_knn = fitcknn(train_examples, train_labels , 'NumNeighbors',
3)

    %Using the trained classifier to predict using the
'test_examples'
    predictions = predict(m_knn, test_examples);




%Evaluate classifiers performance
    % <my comment removed>:
    [c, order] = confusionmat(test_labels, predictions)

    %Create and display a confusion chart using 'test labels' and
predictions
    confusionchart(test_labels, predictions)

    % <my comment removed>:
    p = sum(diag(c)) / sum(c(1:1:end))
```

6. CNN-based features

i) ResNet50 CNN

```
%Relatively accurate 'out the box' using 'MerchData'


%Load Data
%Create Datastore
    unzip('MerchData.zip');
    imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
    [imdsTrain,imdsValidation] = splitEachLabel(imds,0.7);


    %Load pretrained network
```

```
    net = googlenet;


    %Display visualisation of the network and it's layers
    analyzeNetwork(net)


    net.Layers(1)


    inputSize = net.Layers(1).InputSize;


    %Replace final layers
    lgraph = layerGraph(net);


    [learnableLayer,classLayer] = findLayersToReplace(lgraph);
    [learnableLayer,classLayer]


    numClasses = numel(categories(imdsTrain.Labels));


if isa(learnableLayer,'nnet.cnn.layer.FullyConnectedLayer')
    newLearnableLayer = fullyConnectedLayer(numClasses, ...
        'Name','new_fc', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);

elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer')
    newLearnableLayer = convolution2dLayer(1,numClasses, ...
        'Name','new_conv', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);
end


lgraph = replaceLayer(lgraph,learnableLayer.Name,newLearnableLayer);


newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,classLayer.Name,newClassLayer);


figure('Units','normalized','Position',[0.3 0.3 0.4 0.4]);
plot(lgraph)
ylim([0,10])


%Freeze initial layers
layers = lgraph.Layers;
connections = lgraph.Connections;


layers(1:10) = freezeWeights(layers(1:10));
```

```
lgraph = createLgraphUsingConnections(layers,connections);


%Train Network
pixelRange = [-30 30];
scaleRange = [0.9 1.1];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange, ...
    'RandXScale',scaleRange, ...
    'RandYScale',scaleRange);
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter);


augimdsValidation =
augmentedImageDatastore(inputSize(1:2),imdsValidation);


miniBatchSize = 10;
valFrequency = floor(numel(augimdsTrain.Files)/miniBatchSize);
options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',6, ...
    'InitialLearnRate',3e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',valFrequency, ...
    'Verbose',false, ...
    'Plots','training-progress');


net = trainNetwork(augimdsTrain,lgraph,options);


%Classify validation images
[YPred,probs] = classify(net,augimdsValidation);
accuracy = mean(YPred == imdsValidation.Labels)




idx = randperm(numel(imdsValidation.Files),4);
figure
for i = 1:4
    subplot(2,2,i)
    I = readimage(imdsValidation,idx(i));
    imshow(I)
    label = YPred(idx(i));
    title(string(label) + ", " + num2str(100*max(probs(idx(i),:)),3)
+ "%");
end
```

ii) AlexNet CNN

```
%Alexnet is a convolutional neural network that is 8 layers deep
%Very accurate out the box using "MerchData"


%Create image data store
    imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');


%Split up the traing and testing data
    [imdsTrain,imdsValidation] =
splitEachLabel(imds,0.7,'randomized');


    numTrainImages = numel(imdsTrain.Labels);
    idx = randperm(numTrainImages,16);
    figure
    for i = 1:16
        subplot(4,4,i)
        I = readimage(imdsTrain,idx(i));
        imshow(I)
    end

%Setup and using alexnet convolutional neural network
    net = alexnet;

    %Use analyzeNetwork to display the network architecture
information about the network layers.
    analyzeNetwork(net)


    %image input layer using specific size images.
    inputSize = net.Layers(1).InputSize

    %Extracting all of the layers except for the last three
    layersTransfer = net.Layers(1:end-3);

    numClasses = numel(categories(imdsTrain.Labels))

    %Replace the last three layers with a fully connected layer, a
softmax layer, and a classification output layer.
    layers = [
        layersTransfer

fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20,'BiasLearn
RateFactor',20)
        softmaxLayer
        classificationLayer];

%Requires images of size 227 by 227 by 3
```

```
    pixelRange = [-30 30];
    imageAugmenter = imageDataAugmenter( ...
        'RandXReflection',true, ...
        'RandXTranslation',pixelRange, ...
        'RandYTranslation',pixelRange);
    augimdsTrain =
augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
        'DataAugmentation',imageAugmenter);

    %Automatically resize the validation images without data
augmentation
    augimdsValidation =
augmentedImageDatastore(inputSize(1:2),imdsValidation);

%Training using the files from the merchdata set
%Specifying the training options
%Fast learning in new layers and slower learning in the others
%'epoch' = full training cycle using the entire training data set


    options = trainingOptions('sgdm', ...
        'MiniBatchSize',10, ...
        'MaxEpochs',6, ...
        'InitialLearnRate',1e-4, ...
        'Shuffle','every-epoch', ...
        'ValidationData',augimdsValidation, ...
        'ValidationFrequency',3, ...
        'Verbose',false, ...
        'Plots','training-progress');


    %Training the network consisting of transferred and new layers.
    %Opens in a new window - realtime
    netTransfer = trainNetwork(augimdsTrain,layers,options);


%Classifying the validation images using the network
    [YPred,scores] = classify(netTransfer,augimdsValidation);

    idx = randperm(numel(imdsValidation.Files),4);
    figure
    for i = 1:4
        subplot(2,2,i)
        I = readimage(imdsValidation,idx(i));
        imshow(I)
        label = YPred(idx(i));
        title(string(label));
    end


%Checking Accuracy
    %Display four validation images with predicted labels.
    YValidation = imdsValidation.Labels;
    accuracy = mean(YPred == YValidation)
```

KNN

```
%Check and load data
    if ~exist('./MerchData', 'dir')
        error('MerchData does not exist');
    end



% load the dataset
    data = readcell('./Coursework/MerchData.csv');

    % <my comment removed>:
    data(1:1:10, 1:1:end)
    data(1, :) = [];

    % <my comment removed>:
    data = data(randperm(size(data,1)), :);




%separating training and testing data
    rng(0); % please leave this re-seeding of the random number
generator in place so we can compare results
    %
    nTest = round(0.4 * size(data,1))
    data_test = data(1:1:nTest, :);
    data_train = data(nTest+1:1:end, :);



%Creating labels
    %Setting the label index to 2
    label_index = 2;
    %Populate 'test_labels' with a categorical containing test data
and labels
    test_labels = categorical(data_test(:, label_index));
    test_examples = cell2mat(data_test(:, 1:end~=label_index));
    %Populate 'train_labels' with a categorical containing test data
and labels
    train_labels = categorical(data_train(:, label_index));
    train_examples = cell2mat(data_train(:, 1:end~=label_index));



% training the knn classifier

    m_knn = fitcknn(train_examples, train_labels)

    m_nb = fitcnb(train_examples, train_labels)
```

```
    m_dt = fitctree(train_examples, train_labels)

    m_ann = fitcnet(train_examples, train_labels)

    m_svm = fitcecoc(train_examples, train_labels)



%use trained classifier for prediction
    %Using the trained classifier to predict using the
'test_examples'
    predictions = predict(m_knn, test_examples);

    %Using the trained classifier to predict using the
'test_examples'
    predictions1 = predict(m_nb, test_examples);

    %Using the trained classifier to predict using the
'test_examples'
    predictions2 = predict(m_dt, test_examples);

    %Using the trained classifier to predict using the
'test_examples'
    predictions3 = predict(m_ann, test_examples);

   %Using the trained classifier to predict using the
'test_examples'
    predictions4 = predict(m_svm, test_examples);



%Evaluate classifiers performance
    [c, order] = confusionmat(test_labels, predictions)

    %Create and output a confusion chart using the test_labels and
predictions
    confusionchart(test_labels, predictions)

    confusionchart(test_labels, predictions1)


    confusionchart(test_labels, predictions2)


    confusionchart(test_labels, predictions3)


    confusionchart(test_labels, predictions4)


    %Output accuracy
    p = sum(diag(c)) / sum(c(1:1:end))
```

Functions (in order of use):
knn fit
knn calculate distance
knn predict

get brightness - fin
image to gray -
get edges
image gradient xy

get hogs
image gradient
extract hog features

get bag
get words

```
function b = get_brightness(im)
    size(im)
    x = size(im)

    n = x(1,2) * x(1,2)

    %Output the image
    imshow(im);
    %calculate average brightness of an image
    brightness = mean2(im)
    imageArray = []
    imageArray = im
    summedImage = sum(imageArray(),"all")
    finbri = summedImage / n
    b = finbri
end
```

```
function edges = get_edges(im)
    edges = [0 0];

    surf(im, 'EdgeColor', 'none');

    im = im2gray(im);
    imshow(im);
    [Gx, Gy] = imgradientxy(im, 'Prewitt');

    abs(Gy);
    abs(Gx);
```

```
    Gy(Gy>45)
    Gx(Gx<45)


    imshow(Gy)
    imshow(Gx)

    Gy1 = sum(Gy);
    Gx1 = sum(Gx);


    edges = [Gy Gx];
end



function h = get_hogs(im)
    h = [];


    [Gx, Gy] = imgradientxy(im, 'Prewitt');
    [Gmag, Gdir] = imgradient(Gx, Gy);

    imagesc(Gmag);
    imagesc(Gdir);

    CELLSIZE = [16 16];
    [h, v] = extractHOGFeatures(im, 'CellSize', CELLSIZE, ...
    'BlockSize', [floor(size(im,1)/CELLSIZE(1)) floor(size(im,2)/
CELLSIZE(2))], ...
    'UseSignedOrientation', true);

    imshow(im);
    hold('on')
    v.plot();
    h

    hold("off")%bar chart will not show without this
    bar(h)
end



function words = get_words(H)
    rng(0);


    words = [];
    bag= [];


    words = bagOfFeatures(training);
```

```
    im = training.readimage(1);
    im = im2gray(im);
    bag = words.encode(im);


    words = sum(bag)
end



%Extension Functions
    function im_g = my_im2gray(im)

        im_g = [];

        img = im; % RGB image
        %img = ind2rgb(im,map);
        img = im2double(img); % convert image to double datatype
        r = img(:,:,1);
        g = img(:,:,2);
        b = img(:,:,3);

        %gray_image; % image you get after adding three channels
        gray_image = im2uint8(gray_image);

        im_g = gray_image
    end



    function [Gmag, Gdir] = my_imgradient(Gx, Gy)


    Gmag = [];
    Gdir = [];



    end
```