Queen's University Belfast

CSC3061 - Video Analytics and Machine Learning

Group 8

Pedestrian Detection

Bailey Eccles
40173847
beccles01@qub.ac.uk

Wenyong Ye
40247382
wye05@qub.ac.uk

Jack Fullerton
40181939
jfullerton06@qub.ac.uk

April 2, 2020

Contents

1 Introduction

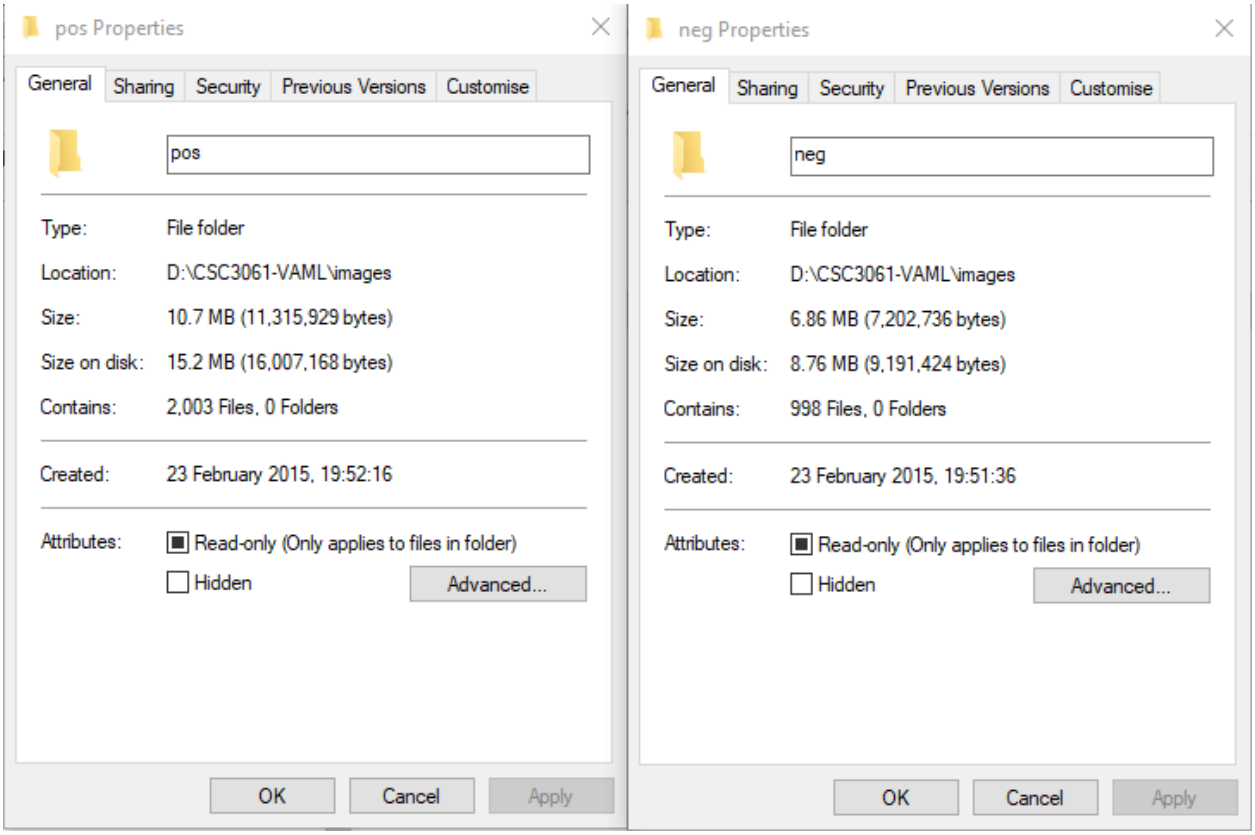
In this project we will train a model to detect pedestrians and develop a system for detection. We will evaluate various pre-processing techniques, feature descriptors and classifiers (along with parameter tuning in each section) to quantitatively justify the best model. We will then test the model against an unknown data set of pedestrians using a multi-scale sliding window detector to supply the model with crops of the environment. The model will make a prediction for each crop. If the crop is detected as a pedestrian then a green bounding box will display on that area. Non-maxima suppression will be used to find the best bounding boxes. The detector will then output a video with the results.

2 Initial Problem Analysis

In supervised classification problems it is important that the training/testing data is balanced in proportion to the number of classes. Imbalanced data can cause machine learning models to favour majority classes which introduces over-fitting to the model and accuracies will be misleading when predictions are performed on an unknown data set.

In pedestrian detection the problem is binary as our models will attempt to classify if a given image contains or does not contain a pedestrian. Therefore we expect the training/testing data set to contain a 50/50 split in positive (presence of a pedestrian) and negative (no pedestrian) images.

For this project the training/testing data set has been provided to us. In the provided folder the number of positive and negative images for training/testing is as follows:



The provided training/testing data set is imbalanced. There appears to be 2003 positive observations (**pos**) and 998 negative observations (**neg**) roughly equaling a 2:1 ratio of positive to negative observations. Upon further inspection there appears to be data duplication in the negative observations:

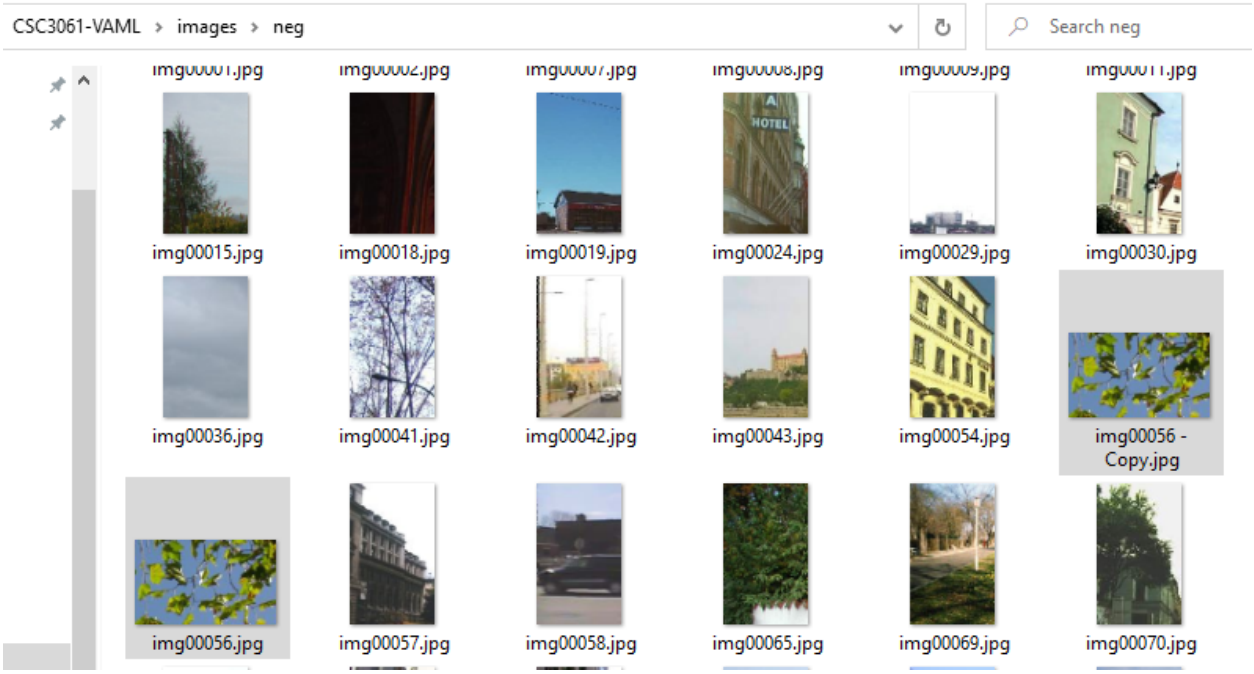
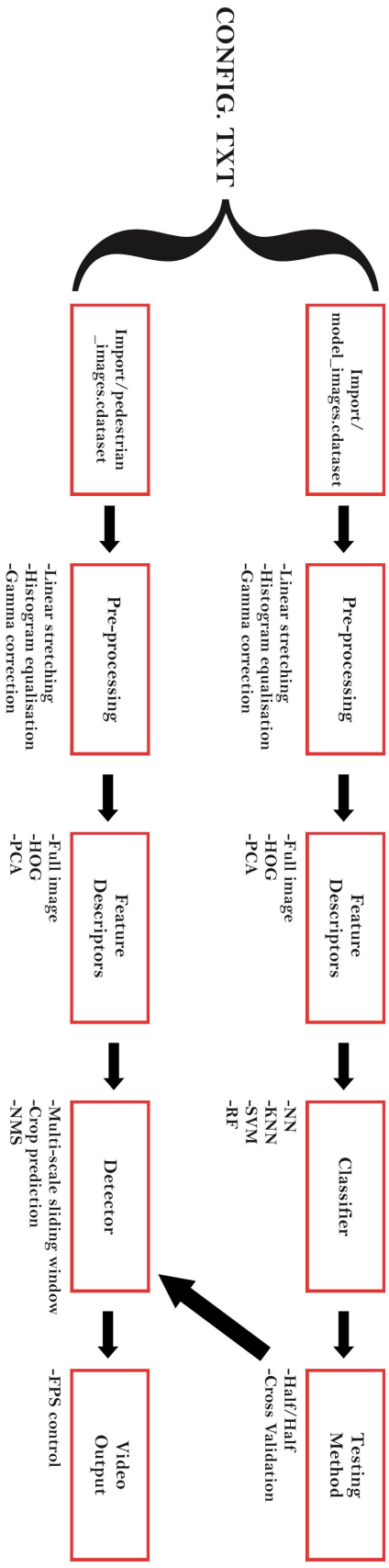


image00056 appears twice in the negative observation data as a duplication anomaly. This will cause the particular observation to be more heavily weighted than other observations during the learning stage for the models. To avoid any of the models over-valuing that particular observation (and any problems that could arise from that) the duplicate entry will be removed from the training/testing data set.

Therefore our training/testing data set consists of 2003 positive observations and 997 negative observations. As discussed previously, imbalanced data can lead to misleading ac curacies during validation and prediction on unknown data (i.e. actual pedestrian detection). Therefore, for evaluation, we will be using an extended suite of metrics to analyse the performance of our models rather than just accuracy to identify any over-fitting issues from the imbalanced data. Metrics such as **recall, precision, specificity, F measure and false alarm rate** will be used in evaluation. Receiver operating characteristic (**ROC**) curves will be included too in our analysis. Our aim is to use quantitative analysis on these metrics to provide empirical evidence in determining the best model for pedestrian detection.

We will also use the opportunity to train/test our models on the full **imbalanced** data set (2003 positive, 997 negative) and a **balanced** data set (997 positive, 997 negative) and compare and contrast the effect on the models.

3 Developed Pipeline Overview



4 Pre-processing

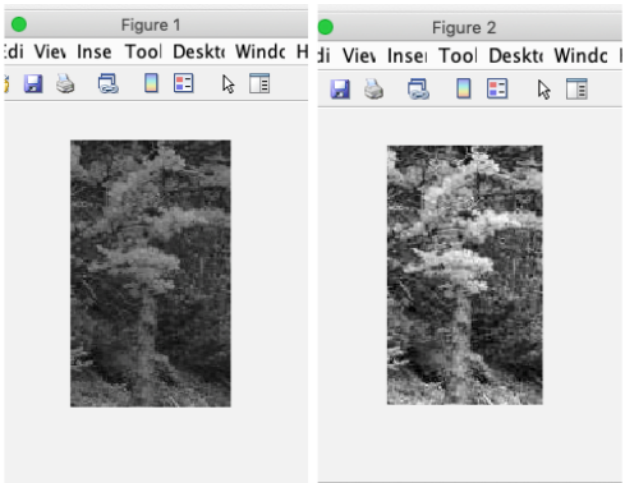
4.1 (Automatic) Linear Stretching

Linear stretching is widely used in the field of contrast enhancement. The straight line equation is used to re-distribute the values of the input map with wider dynamic range from 0 to 255. Considering noisy pixels exist in the image, we develop the advanced linear stretching that can remove noisy pixels automatically before performing linear stretching. The first and last 10 pixels are considered as noisy pixels.

```
function images = enhanceContrastALS(images, num_row, num_col)
    %linear stretching without user input
    %automatically generate parameter m,c

    %Iin: input image
    %noise: number of noise pixels to be ignored

    %define the first and last 10 pixels as noise
    noise=10;
    %retrieve each image
    for i = 1:size(images,1)

        Iin_vector = images(i,:);
        Iin_reshape = reshape(Iin_vector,num_row,num_col);
        Iin = uint8(Iin_reshape);
        %Sort image pixels in ascending order
        sorted=sort(Iin(:));
        sorted=double(sorted);

        %min and max value used for linear stretching
        minVal = sorted(noise + 1);
        maxVal = sorted(size(sorted,1) - noise);
        %calculate parameters
        coe=polyfit([minVal,maxVal],[0,255],1);
        %generate look-up table
        Lut = contrast_LS_LUT(coe(1),coe(2));
        Iout = intlut(Iin,Lut);
        Iout_reshape = reshape(Iout,1,num_row * num_col);
        %output images
        images(i,:) = Iout_reshape;
    end
end
```

```
function Lut = contrast_LS_LUT(m,c)
%Look-up table for linear stretching
array=zeros(1,256);
%straight line equation
for i=0:255
    if i < -c/m
        Lut = 0;
    elseif i > (255 - c)/m
        Lut = 255;
    else
        Lut = m*i+ c;
    end

   array(i+1)=Lut;
end
%return look-up table
   Lut=uint8(array);
end
```

The figures above illustrate the implementation of advanced linear stretching. The function 'enhancedContrastALS' generates and returns the images after applying linear stretching. The function sorts image pixels' value in ascending order and removes the first and last 10 pixels which are considered as noise. The MATLAB function polyfit is used to provide best fit of coefficients for the polynomial. The look-up table(LUT) is created based on the calculated coefficients, and applied to all valid pixels of images.



The figure above is the comparison of images before or after applying linear stretching. The image after linear stretching has higher contrast. The dynamic range of the image becomes wider, but some noisy pixels will be more obvious in the high frequency region. We can use the images after pre-processing and explore its effect on classification and detection.

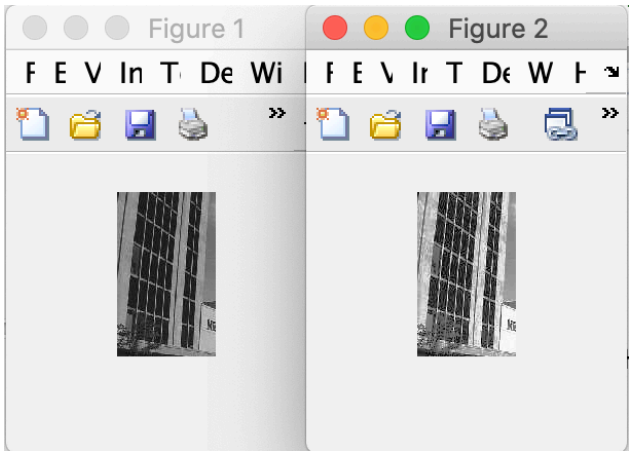
4.2 Histogram Equalisation

Histogram equalisation is a computer image processing technique used to improve the contrast of images. This improvement can lead to more accurate results from the feature extractor/classifier. Histogram equalisation achieves this by spreading out the more frequently used pixel values...this results in a larger intensity range in the image.

```
function Lut = contrast_HE_LUT(Iin)
    %% Init array: array
    array=zeros(1,256);
    %% Gen Cumulative Histogram: ch
    histogram = imhist(Iin);
    ch = cumsum(histogram);
    %% Gen LUT using transfer function for each pixel value: Lut
    for i=0:255
        Lut = max(0,round((256*ch(i+1)/numel(Iin)) - 1));
        array(i+1) = Lut;
    end
    %% Return: Lut
    Lut=uint8(array);
end
```

```
function images = enhanceContrastHE(images, num_row, num_col)

    for i = 1:size(images,1)
        Iin_vector = images(i,:);
        Iin_reshape = reshape(Iin_vector,num_row,num_col);
        Iin = uint8(Iin_reshape);
        %% Gen LUT: Lut
        Lut = contrast_HE_LUT(Iin);

        %% Apply LUT to Iin and return: Iout
        Iout = intlut(Iin,Lut);

        Iout_reshape = reshape(Iout,1,num_row * num_col);

        images(i,:) = Iout_reshape;
    end
end
```

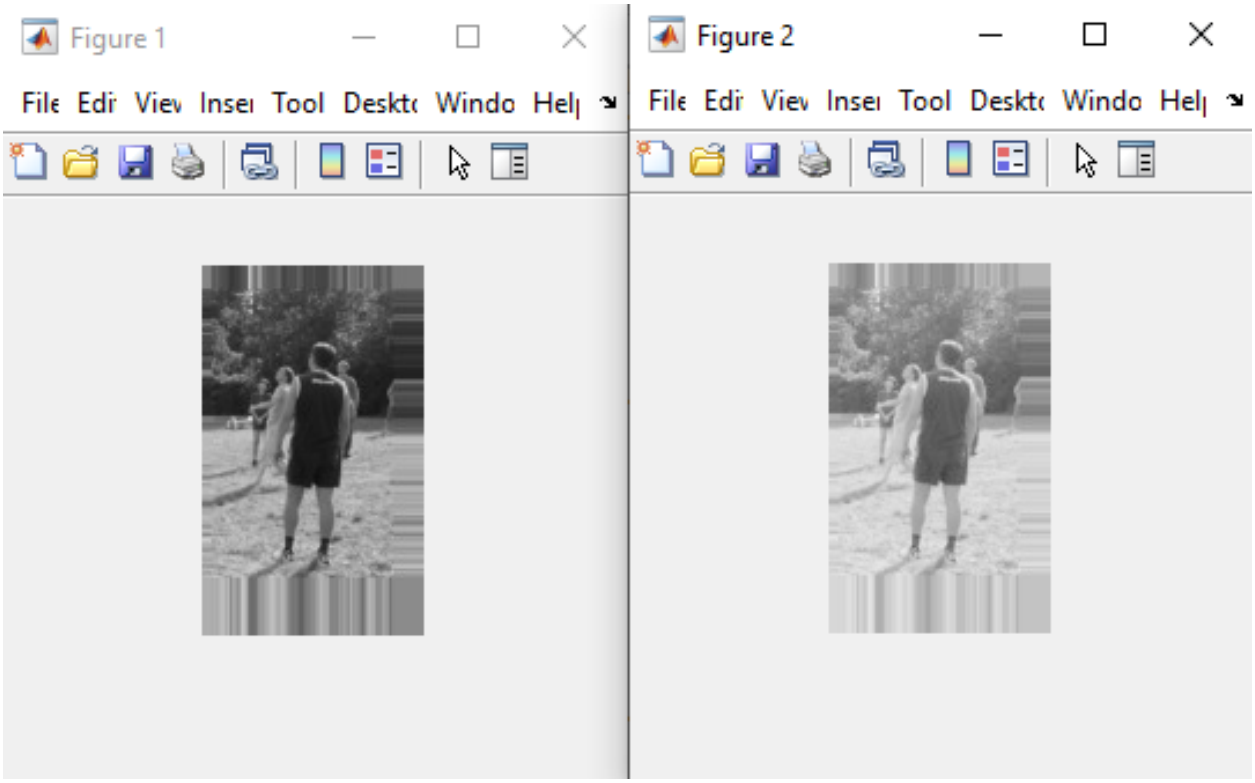
In order to implement histogram equalisation the above functions were created. The first one generates a cumulative histogram from the image, then **generates a look up table and returns all 256 updated pixel values**. This LUT is then applied to all images using the above function called 'enhanceContrastHE'.

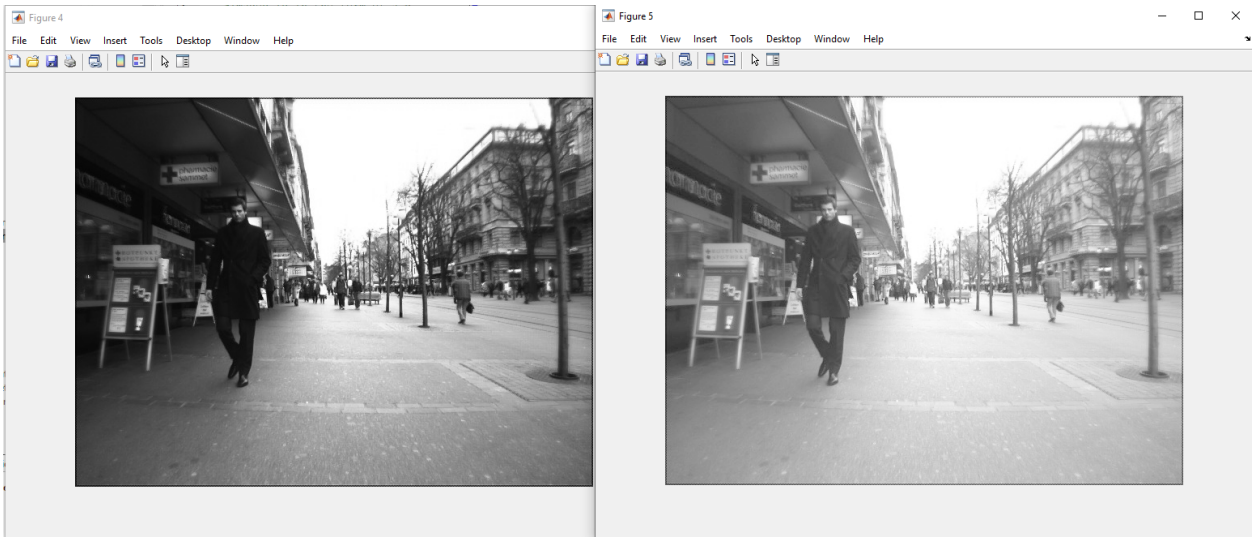


here you can see the before and after from histogram equalisation. We can see that the contrast in the second picture has improved making it more useful for feature detection and classification.

4.3 (Automatic) Gamma Correction

Gamma correction is commonly used to correct the contrast of an encoded image (due to the linear capture of exposure in camera sensors). This pre-processing technique may be useful to help provide more clarity in feature extraction. We will use the MatLab function *lin2rgb* (https://uk.mathworks.com/help/images/ref/lin2rgb.html) that automatically applies gamma correction to a given image. Automatic gamma correction is preferred as each image may require different amounts of correction, therefore, a function that provides an automatic estimation of correction for each case should be used. Below are an example of Automatic Gamma Correction on the training/test data set and the pedestrian data set. Left is without Gamma Correction, right is with Gamma Correction.





In the above images the Automatic Gamma Correction pre-processing made pedestrians, in particularly dark regions, brighter and more distinguishable by eye. This effect may be beneficial during feature extraction for when the models learn the training/testing data set and when the models perform detection on the pedestrian data set. We will explore the effect of this pre-processing technique quantitatively on various models and with various feature descriptors. We will also explore the effect of this pre-processing technique during the pedestrian detection phase. When this pre-processing technique is used in evaluation the label **AGC** will be used.

5 Feature Descriptors

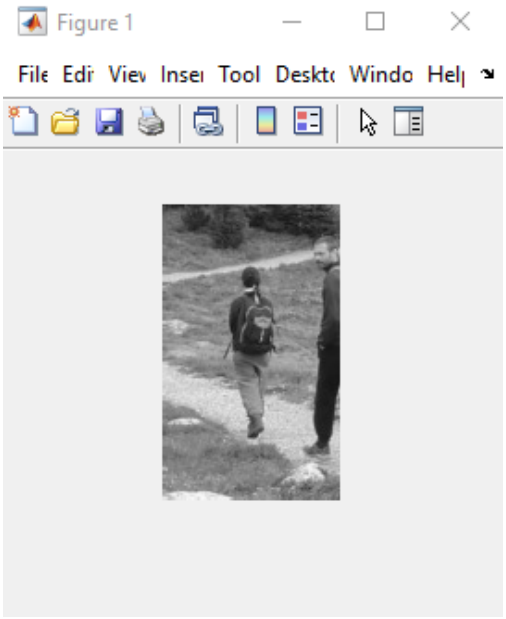
5.1 Full Image

A full image feature descriptor encodes the entirety of the raw pixels as a feature vector. No specific regions of interest are extracted from the data unlike other feature descriptors. Therefore the feature vector size is expected to be relatively larger than other feature descriptors (the size of height x width as the entirely of the image is encoded to the feature vector). The larger feature vector size has multiple downsides.

Firstly the computation cost is greater since the model has to learn from the entirety of the image. Storing the entirety of the full image in memory is also greater than other feature descriptors that simply encode and store specific regions of interest.

Secondly, the classification performance of this feature descriptor is expected to be worse for machine learning than almost all other feature descriptors. The full encoding of an image to a vector includes all regions of the image, this includes noise and uninteresting regions that do not represent what we want our learning methods to classify. A learning method may use this information and learn erroneous patterns that lead to misclassification.

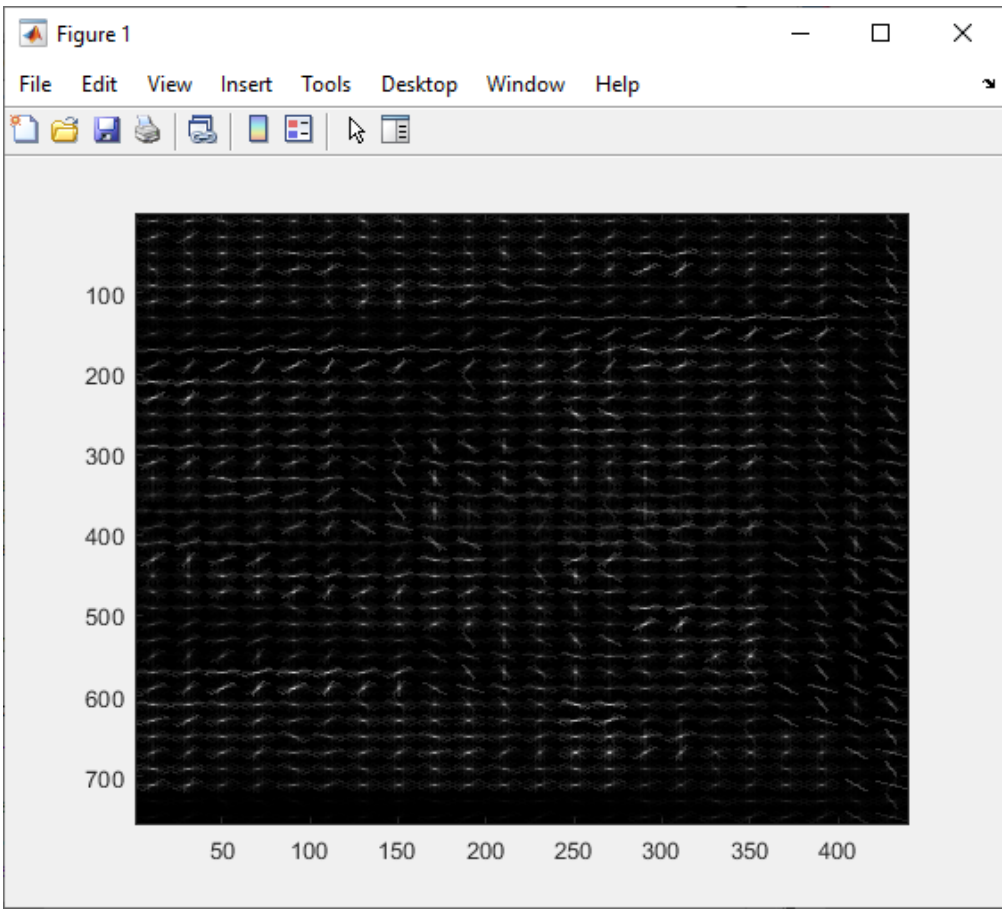
Although expected to not be a great feature descriptor, we will still test the effectiveness of this feature descriptor against the others to quantitatively evaluate it's performance. Below is a visual representation of a full image feature descriptor. It looks and is exactly the same as the a pre-processed image, however, it is to illustrate that no extraction is performed and how it visually compares to other feature descriptors such as HOG.



5.2 Histogram of Oriented Gradients (HOG)

Histogram of Oriented Gradients (HOG) is a feature descriptor specifically designed for pedestrian detection. HOG tries to tries to capture the shape of structures (pedestrians, animals etc.) in images by calculating gradients of localised areas. Each area is represented as a cell. HOG implementations also include features such as overlapping local contrast normalization that improves accuracy.

We will implement HOG in this project using the HOG toolbox provided in the practicals. This HOG implementation produces feature vectors that are approximately half the number of dimensions as a full image feature descriptor (7524 dimensions instead of 15360) other implementations can use different bin sizes and may produce feature vectors of bigger or smaller size. From the smaller and more refined feature vector we can expect HOG to perform better for the models than the full image feature descriptor as this algorithm specifically extracts areas of interest and patterns for the learning methods. The smaller vector space will also provide lower memory and CPU costs when training and testing the models. However, HOG will also extract noise as shown in the outer regions in the visualisation below. Below is a visual representation of a HOG feature descriptor.



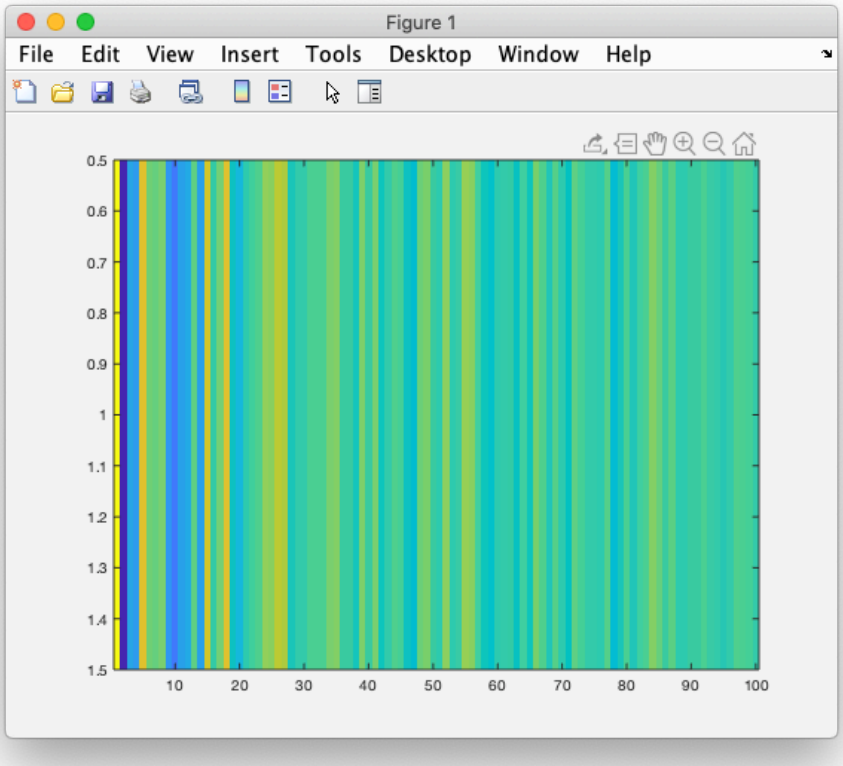
5.3 Principal Component Analysis (PCA)

The principal component analysis (PCA) is chosen as feature descriptor is because it can reduce the dimensions of data-set, and retain the main features. Unlike the full image, the PCA can obtain the interesting regions, and pass the result to classifier for training. This method improves the performance and accuracy of classifier to some extent.

In this assignment, the PCA is used for reducing dimensions of original image data-set. The PCA function can reduce the dimensions into given value. The PCA function obtains principle components to reduce calculation, and for most situations, it need to cover up to 95% of original content. The disadvantages of PCA are time consuming, and it takes much computing resources. Therefore, the process for running the PCA takes much time to extract main features. If the data-set contains many features, the efficiency will be much lower than full image or HOG.

The PCA is used as the feature descriptor, it can also be used in the section of detection. If the PCA is used in detection, all sliding window images of each frame need to perform PCA for one time. Considering there are many frames of the whole video, the PCA is not the best choice for detection because of limited computational resources. It is less useful than HOG in the sliding window detection system, and will only be used as the feature descriptor. It is optional to run PCA on the detection system.

The figure below shows the PCA result with dimensions reduced to 100. The X-axis indicates there are 100 dimensions and the Y-axis indicates the value of each dimension.



6 Models for classification

6.1 Nearest Neighbour (NN)

The nearest neighbour(NN) is one of the forms of K-nearest neighbour, whose K equals to 1. The NN is one of the supervised learning algorithms used for classification, which requires labels for learning. The NN is used for testing the special cases that finding nearest neighbouring data point in given images. For calculating the distance between data points, there are many ways such as Euclidean distance and Mahalanobis distance. Euclidean distance is used and implemented in the K-NN function.

The implementation of NN and KNN are the same, the only difference is the input value of K. The implementation details is illustrated in the KNN (section 6.2).

The performance of NN will be evaluated in the part of evaluation according to the classification results. It will also be compared with K-NN with different value of K.

6.2 K-nearest Neighbour (k-NN)

The K-nearest neighbour (K-NN) is a supervised learning method for classification. The K-NN aims at finding K nearest neighbouring data points and voting for class of labels. The value K of K-NN usually uses odd number and some common values are K=3,5,7... If the K is too small, it will more likely to be over-fitting. In contrast, if the K is too latrge, it will more likely to be under-fitting. Over-fitting means the classifier is too closely fit the training data-set, and under-fitting means the model cannot obtain relationship between data samples inside the data-set. The chosen value of K can influence the performance of classifier, so the proper value of K need to be selected through experiment.

The implementation details are illustrated as follow.

```
function modelKNN = KNNTraining(images, labels)
%trainning function for KNN
%images: images
%labels: labels of images
modelKNN.neighbours=images;
modelKNN.labels=labels;

end
```

The figure above shows the training function of K-NN. The function input images and corresponding labels, and assign the values to the model.

```
function prediction = KNNTesting(testImage, modelKNN, K)
   %KNN testing
   %testImage: image for testing
   %modelKNN: model
   %K: value of K
   %n0 is negative, n1 is positive
   n0=0;
   n1=0;
   array=[];
   %calculate Euclidean Distance
   for i=1:size(modelKNN.neighbours,1)
       array(i)=EuclideanDistance(testImage,modelKNN.neighbours(i,:));
   end
```

As seen in the above figure, the K-NN testing function has three input parameters: testing image, K-NN model and K value. The Euclidean distance is calculated and the results are stored in an array. A for loop is used to compute the Euclidean distance between the testing image and each image in the model.

```
function dEuc=EuclideanDistance(sample1, sample2)
    %Euclidean Distance
    dEuc=sqrt(sum((sample1-sample2).^2));
end
```

The Euclidean distance is calculated between each sample. The parameters needed are the (x,y) on coordinate axis. After calculation, the results are output to the testing function.

```
    %sort array in ascending order
    [vals, indxs] = sort(array);

    %Check if K is larger than number of elements
    if size(modelKNN.labels,1)<K
        K=size(modelKNN.labels,1);
    end
    %iterate each element in array
    for j=1:K
         if modelKNN.labels(indxs(j))==1
             n1=n1+1;
         else
             n0=n0+1;
         end
    end
    %define the predicted label of image
         if n0>n1
             prediction=-1;
         else
             prediction=1;
         end
end
```

The figure above shows the code after obtaining Euclidean distance. Sort the array containing Euclidean distance into ascending order. Iterate all elements in the array and record their labels. Count the number of both positive and negative votes and choose the large number as label. Assign -1 as negative and 1 as positive label, then, return the prediction result.

This K-NN function is called from the classifier testing part of pd.m. The results of predicted labels are combined into an array.

6.3 Support-Vector Machine (SVM)

Support-Vector machines (SVMs) are a supervised learning model used for classification and regression. In this project we will be using two implementations of SVM for classification. For the models using Half/Half testing we will be using the supplied SVM Toolbox from the practicals (*SVMTraining* and *SVMTesting*) as it is simple to use and provides the functionality needed for Half/Half Testing. For a cross-validated models we will be using the MatLab *fitcsvm* function as it supplies extra functionality for cross validation and hyperparameter optimisation.

The cross validated SVM models will under-go hyperparamter optimisation to find the optimal cost value for each SVM. The cost value is the maximum penalty imposed on margin-violating observations. Optimising this hyperparameter helps to prevent overfitting. The paramaters *'OptimizeHyperparameters','auto'* will be passed to the models during training to turn on hyperparamter optimisation.

The default kernel option will be used which is *linear* as this kernel option is for two-class classification such as this.

6.4 Random Forest (RF)

Random forests are a large number of decision tress that operate as an ensemble. Each tree makes a prediction and the class with the most votes becomes the model's prediction.

Random forests do not require cross-validation as by design are already cross validated (splitting data amongst random trees). Random forests however perform badly with imbalanced data (as the extra class weight are weighted more heavily in the tree's decisions). It will be interesting to see the differences between the balanced data set and the full imbalanced data set.

We will be using various values for number of trees to find the best random forest model.

7 Testing Methods

7.1 Half/Half

For the testing method Half/Half (HH), it is use for splitting the whole data-set into training data-set and testing data-set. Both two data-sets have the same number of samples.

The process is presented as follow. Firstly, shuffle all samples in the data-set. Divide the data-set into testing data-set and training data-set, ensure the same number of samples of each data-set. We will use Half/Half as testing method to evaluate the model.

7.2 Cross Validation

The cross validation (CV) is one of the common testing methods. The K-fold cross validation is used for testing. The K-fold means the data-set spited into K groups, each group has the same number of samples. For cross validation, choose one group of samples each time as testing data-set, leave the other groups as training data-sets.

The process is presented as follow. First of all, shuffle all samples of the data-set randomly. Then. split the data-set into K groups. For each group, take one group as testing data-set and take other remaining groups as training data-set. For each unique training and testing process, fit a model and have its evaluation on testing set. After the iteration of all groups, summarize the accuracy results and we can get the average evaluation scores of model.

For the configuration of K, it is common to choose K=5 and K=10. Different value of K can have influence on the performance of the classifier. Multiple K-fold cross validation tests are performed using different K value in later sections.

8 Model Evaluation - k-NN

8.1 Half/Half NN - Full Image

K-NN with K=1 (NN)

Method	Data Set	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
-	Balanced	81.2	55.6	83.6	94.4	66.8	5.6	57.6
ALS	Balanced	67.3	5.9	71.4	98.8	10.9	1.2	16.0
HE	Balanced	84.3	74.0	78.6	89.7	76.2	10.3	69.7
AGC	Balanced	66.1	0	NaN	1	0	0	11.1
-	Imbalanced	65.3	50.0	96.1	96.0	65.8	0.04	72.8
ALS	Imbalanced	73.8	66.7	91.7	88.0	77.2	12.0	77.1
HE	Imbalanced	77.5	71.8	92.9	89.0	81.9	11.0	80.2
AGC	Imbalanced	65.6	49.5	97.8	97.8	65.7	2.2	73.6

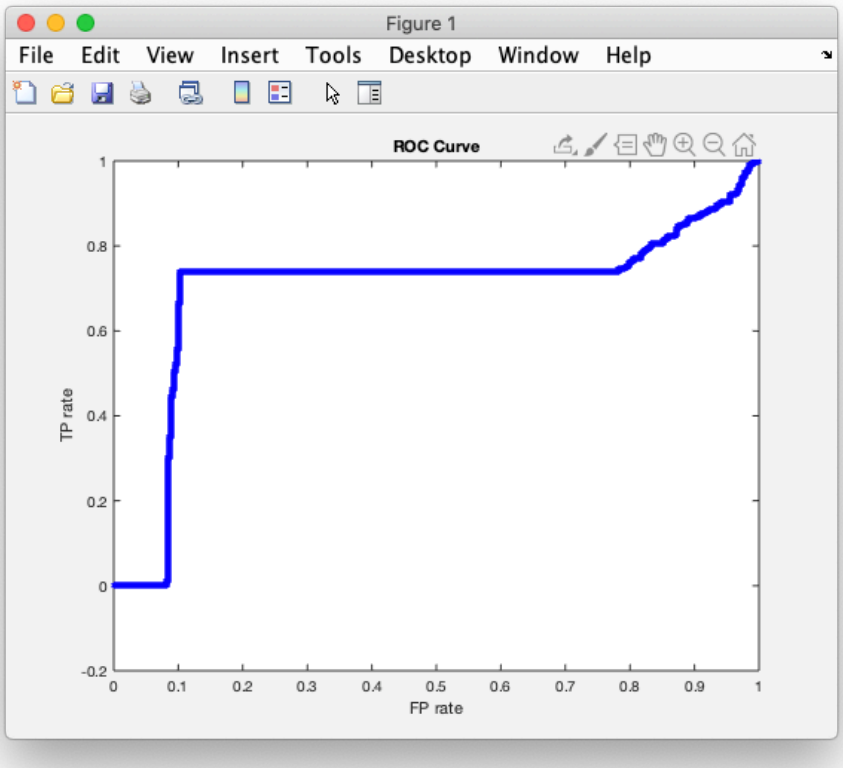
Table 1: Pre-processing effect on HH-NN-Full Image(%)

The HH–NN-Full Image model has the best performance with the **full imbalanced data-set** and the **Histogram Equalisation** according to measurements of both accuracy and AUC.

As the table illustrated above, the NN is performed on both **balanced** and **imbalanced** data-sets. Three different kinds of algorithms for pre-processing are used and having comparisons. The expected result is the imbalanced data-set will perform better than that of the balanced data-set. However, the result of NN using **AGC** and **balanced data-set** runs into problem. The measurement parameters are 0 and the AUC is low. The value of the parameters is much different from other results. Result generated by AGC and balanced data-set is unreliable and we will omit it.

The Histogram Equalisation allows the accuracy increased by 12.2% (65.3->77.5). The AUC improvement is 7.4%(72.8 -> 80.2).

The figure below is about the ROC for H/H NN-Full Image data-set with imbalanced data and Histogram Equalisation.



The ROC figure shows that the performance of classifier is good because of covering up most under curve area. We will retain the result of model using **HE** and **imbalanced data-set**, to have comparison with other models in next sections.

8.2 Half/Half NN - HOG

K-NN with K=1 (NN)								
Method	Data Set	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
-	Balanced	81.2	55.6	83.5	94.3	66.8	5.6	57.6
ALS	Balanced	78.5	69.8	67.8	83.0	68.8	16.9	62.2
HE	Balanced	84.3	73.9	78.6	89.7	76.2	10.3	69.7
AGC	Balanced	82.4	55.9	87.9	96.0	68.3	3.9	59.1
-	Imbalanced	65.3	50.0	96.1	96.0	65.8	0.04	72.8
ALS	Imbalanced	73.8	66.7	91.7	88.0	77.2	12.0	77.1
HE	Imbalanced	77.5	71.8	92.9	89.0	81.9	11.0	80.2
AGC	Imbalanced	65.6	49.5	97.8	97.8	65.7	2.2	73.6

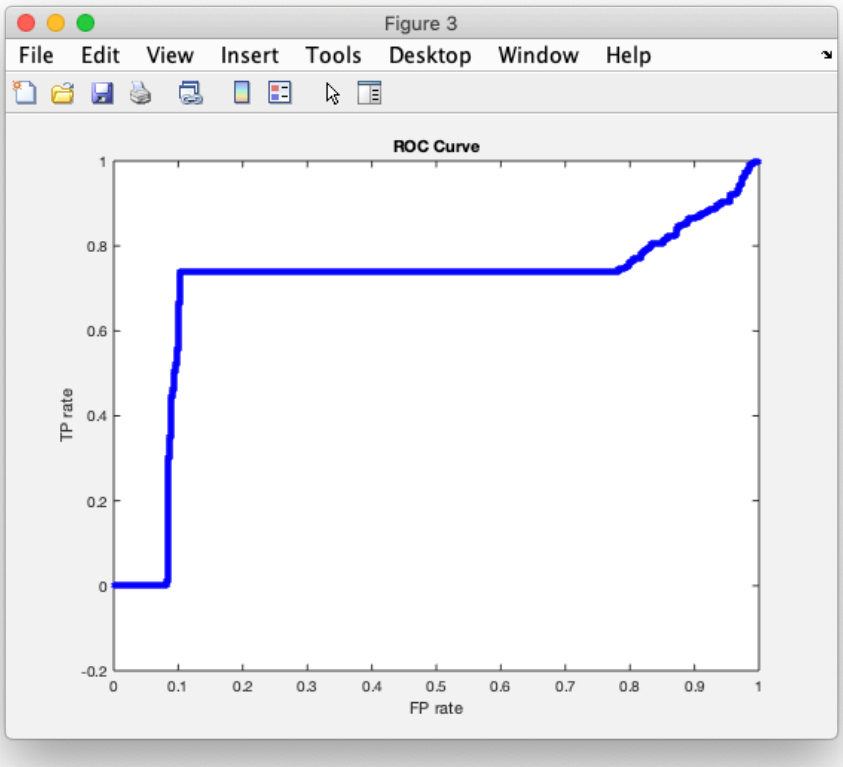
Table 2: Pre-processing on HH-NN-HOG(%)

The **HH-NN-HOG** model has the greatest performance with the **full imbalanced data-set** and the **Histogram Equalisation**, which is the same as the HH-NN-Full Image model.

The table below shows that most of the results are the same as the HH-NN Full Image. The accuracy and AUC are slightly different between two different models. The performance of the imbalanced data-set of this model is the same as the previous model. For the imbalanced data-set, there is an obvious improvement on AUC, which means that the imbalanced data-set can improve the performance of classifier to some extent.

For the measurements of this model, the **Histogram Equalisation** allows the accuracy increased by 12.2% (65.3->77.5). The AUC improvement is 7.4%(72.8 -> 80.2). The value is the same as the Full Image model.

The figure below is about the ROC for H/H NN-HOG data-set with **imbalanced data-set** and **Histogram Equalisation**.



The ROC figure is similar to the HH-NN Full Image model and the curve is almost the same. The performance is good because of covering up most under curve area. We will retain the result of model using **HE**, **HOG** and **imbalanced data**, and have the same parameters setting of K-NN.

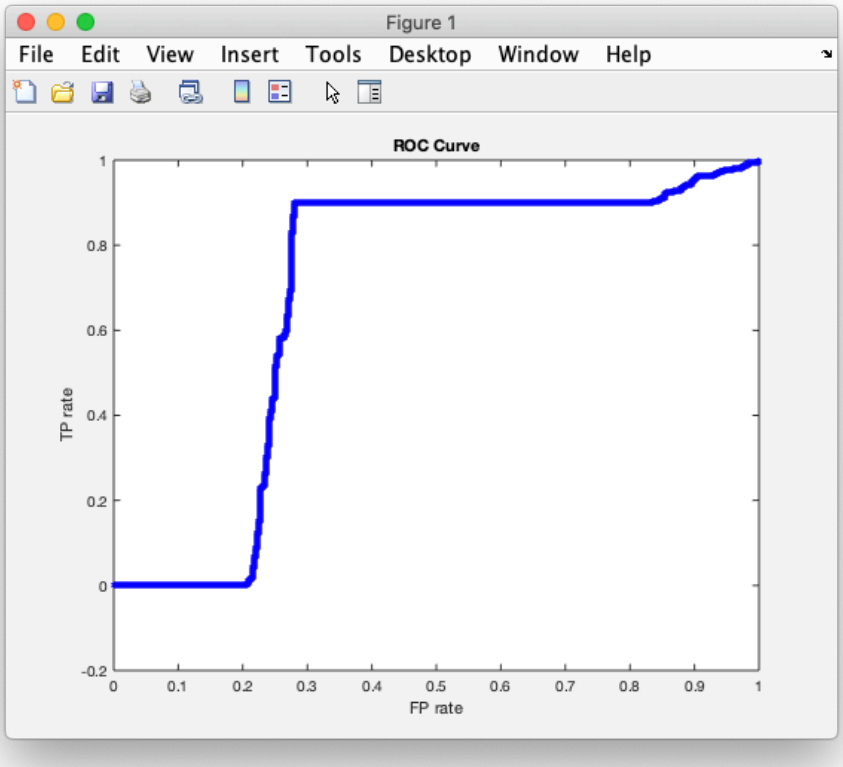
8.3 Half/Half K-NN

	K-NN with different value of K						
K	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
5	84.3	73.9	78.6	89.7	76.2	10.3	69.7
10	78.5	73.4	92.9	88.8	82.0	11.2	80.6
20	80.2	75.4	69.1	82.7	72.1	17.2	65.8
50	84.6	87.6	89.2	78.8	88.3	21.2	82.9
100	74.9	91.2	58.3	66.4	71.1	33.5	64.8
200	71.6	94.0	54.7	60.1	69.2	39.9	61.2

Table 3: Pre-processing on HH-K-NN-Full Image(%)

The table above is about the K-NN using **Histogram Equalisation** and **HOG**. The K value of KNN is listed in the table. The parameter setting for the K-NN is using **HOG**, **imbalanced data-set** and **Histogram Equalisation**, which illustrates best performance in previous section.

When the K=50, the K-NN classifier shows the highest accuracy which is 84.6. Besides, the corresponding AUC is 82.9, and it is the highest too. We can observe that the accuracy of K-NN increases as the K value increasing until the K reaches around 50, which is the highest accuracy. Then, the accuracy decreases after K value greater than 50, this is because of model under-fitting.



The figure above illustrates the ROC of K-NN when K=50. The AUC is 82.9. The trend of this curve is similar to that of NN model. The main difference is that the TP value is higher than that of NN across the x-axis. It means more samples are classified correctly in this range.

In the next sections, we will use K-NN with K=1and K=50. We will make a comparison on the performance of PCA.

8.4 Half/Half K-NN - PCA

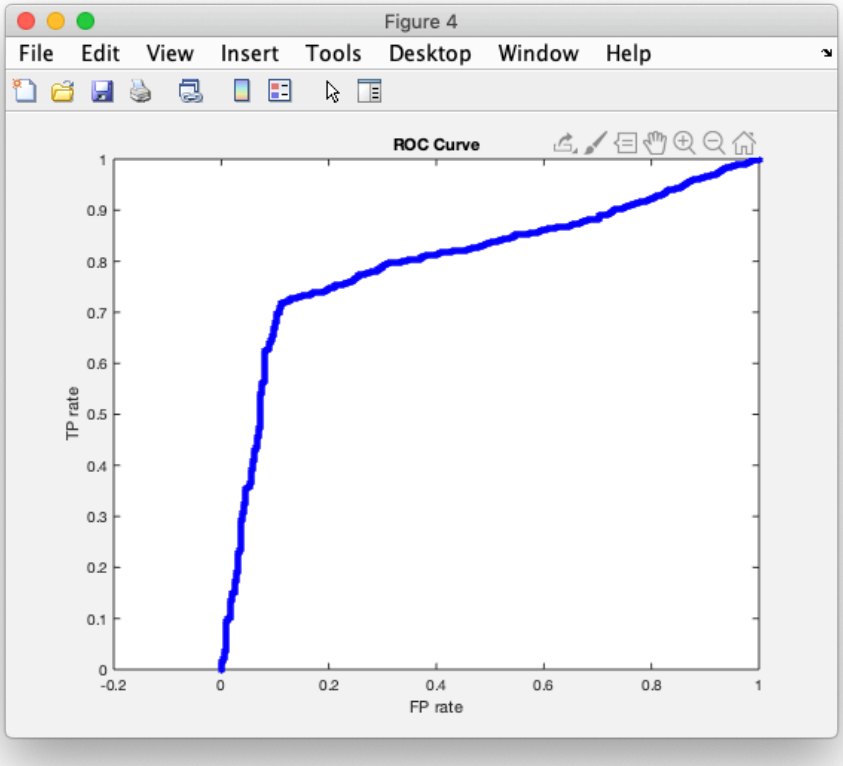
The two tables below show the PCA results using K-NN with K=1 and K=50. The reason is for testing the influence of different value of K on the performance of PCA. We should observe obvious difference through two testings with different value of K.

The parameter settings are **Histogram Equalisation**, **imbalanced data-set**. The only difference is the value of K of K-NN.

K-NN with K=1 (NN)							
ndim	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
default (500)	77.5	71.8	92.8	81.0	81.0	11.0	80.3
100	77.5	71.8	92.9	89.0	80.9	11.0	80.3
1000	77.5	71.8	92.9	89.0	81.0	11.0	80.3

Table 4: Number of dimensions effect on HH-NN-PCA with HE and full imbalanced data set (%)

The figure above is the testing result of NN. The result values are the same and it can be considered that the dimensions of PCA has less influence on its performance. The PCA with dimensions that cover up most components can have the same performance. The default setting of the PCA is chosen by the function automatically, which covers



up 95% components.

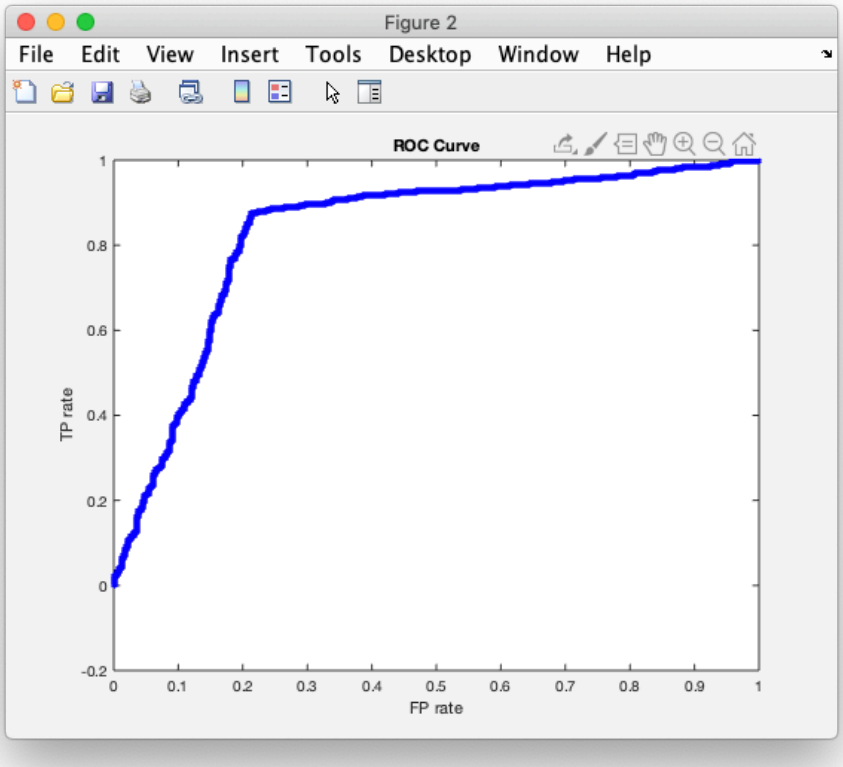
The figure above is about the ROC of PCA using NN. The AUC is 80.3 and it is almost the same as the performance of NN using **Full Image** and **Histogram Equalisation** that is 80.2.

The table above shows the PCA using K-NN with K=50. The results of three different dimensions are the same. Through testing PCA with different K value of K-NN, it can be concluded that the dimensions of PCA have

K-NN with K=50							
ndim	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
default (500)	84.7	87.6	89.2	78.8	88.4	21.2	83.0
100	84.7	87.6	89.2	78.8	88.4	21.2	83.0
1000	84.6	87.6	89.2	78.7	88.3	21.2	82.9

Table 5: Number of dimensions effect on HH-K-NN-PCA with HE and full imbalanced data set (%)

limited influence on the performance of PCA, if including most of its components. The performance of PCA can be improved through choosing appropriate value of K in K-NN. The accuracy is 84.7 and the AUC is 83.0. Compared with the NN, the improvement of accuracy is 7.2 (77.5->84.7), and the improvement of the AUC is 2.7 (80.3->83). The improvement is limited, one of the reasons maybe the dimensions chosen are enough for PCA to extract main



features.

The figure above illustrates the ROC of PCA using K-NN with K=50. The AUC is 83.0. The PCA is unlikely to be a good feature descriptor for the K-NN model because the limited improvement of accuracy. Besides, the PCA is time consuming and takes a lot computational resources.

8.5 Cross-validated K-NN -HOG

There are two tables for cross-validated K-NN-HOG. One with K=1, and another with K=50

K-NN with K=1							
k-fold	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
5	88.0	85.2	96.5	93.8	90.5	6.2	89.4
10	88.3	85.4	96.6	94.0	90.7	6.0	89.5
30	88.4	85.5	96.7	94.2	90.8	5.8	89.7

Table 6: Cross validation k-fold effect on CV-NN-HOG with HE and full imbalanced data set (%)

The table above is **CV-NN-HOG** using k-fold cross-validation for testing. The parameter of the NN is **HOG**, **Histogram Equalisation** and **imbalanced data**. The setting demonstrates the best performance in the previous sections. As the table illustrates, both the accuracy and the AUC increase as the value of fold chosen becoming larger.

Compared the results with HH-NN, the improvement of accuracy is 10.7%(77.5->88.2), the improvement of the AUC is 9.3% (80.2->89.5).

K-NN with K=50							
k-fold	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
5	87.8	84.8	96.4	93.7	90.2	6.3	89.2
10	87.8	84.6	96.7	94.3	90.3	5.7	89.4
30	88.0	85.0	96.7	94.1	90.5	5.9	89.5

Table 7: Cross validation k-fold effect on CV-KNN-HOG with HE and full imbalanced data set (%)

The table above is **CV-KNN-HOG** using k-fold cross-validation for testing. The parameter of the K-NN is **HOG**, **Histogram Equalisation** and **imbalanced data**. The K value of K-NN is set to 50 which has the best performance. The setting demonstrates the best performance in the previous sections. As the table illustrates, both the accuracy and the AUC increase as the value of fold chosen becoming larger, which are the same as that of NN. The accuracy of NN and K-NN here are almost the same.

Compared the results with **HH-K-NN**, the improvement of accuracy is 3.6%(84.6->88.2), the improvement of the AUC is 6.6% (82.9->89.5).

9 Model Evaluation - SVM

9.1 Half/Half SVM - Full Image

Method	Data Set	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
-	Balanced	41.9	52.1	29.7	36.7	37.8	63.3	24.8
ALS	Balanced	56.0	75.1	41.7	46.1	53.6	53.9	40.2
HE	Balanced	63.2	80.8	47.5	54.2	59.8	45.8	49.2
AGC	Balanced	70.2	81.4	54.0	64.4	65.0	35.5	57.0
-	Imbalanced	74.2	79.5	81.4	63.6	80.4	36.4	71.7
ALS	Imbalanced	65.2	75.8	73.0	44.0	74.4	56.0	59.5
HE	Imbalanced	63.7	76.0	71.4	39.0	73.6	61.0	56.9
AGC	Imbalanced	79.0	83.4	84.8	70.2	84.1	29.8	77.4

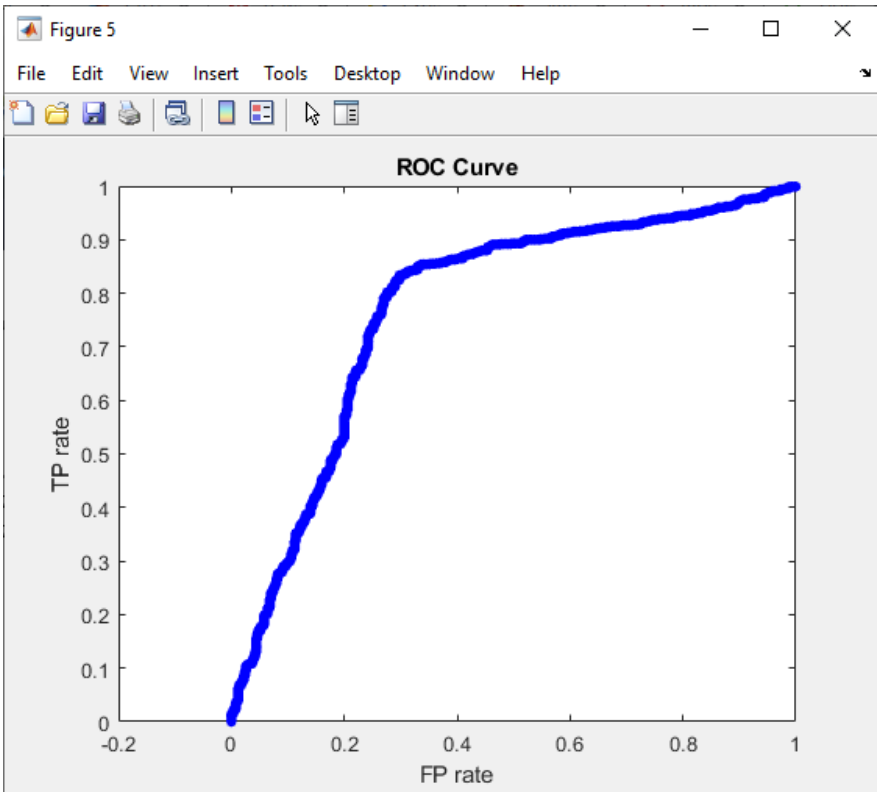
Table 8: Pre-processing effect on HH-SVM-Full Image (%)

The HH-SVM-Full Image model using **Automatic Gamma Correction** and the **full imbalanced data set** appears to have the best evaluation metrics out-performing all the other models displayed above.

Pre-processing was performed on both the balanced data set and imbalanced data set to compare the different effects on the SVM model . We would expected the balanced data set to show a lesser degree of overfitting, however, it appears the larger degree of under-sampling from the balanced data set gives the SVM model a massive performance cost over the imbalanced data set. It appears that a larger, however imbalanced data set, works better for this particular model than a smaller, balanced data set.

The **Automatic Gamma Correction** pre-processing appears to help the SVM model with learning allowing for a 5% better accuracy (74.2 $\rightarrow$ 79.0) and 6% better AUC (71.7 $\rightarrow$ 77.4) compared to the same model without pre-processing.

Below is the ROC for HH-SVM-Full Image with AGC and full imbalanced data set:



Looking at the ROC it shows there is room for much improvement. We will compare this with the other SVM models in a later section. Going forward for HH-SVM-Full Image the model with AGC and full imbalanced data set will be used to compare and contrast with other models.

9.2 Half/Half SVM - HOG

Method	Data Set	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
-	Balanced	95.7	97.3	90.6	94.8	93.9	5.2	93.3
ALS	Balanced	95.5	97.6	89.9	94.4	93.6	5.6	93.1
HE	Balanced	95.4	97.3	89.9	94.4	93.5	5.6	93.1
AGC	Balanced	95.2	97.3	89.4	94.1	93.2	5.9	92.3
-	Imbalanced	96.9	97.7	97.6	95.2	97.7	5.0	96.2
ALS	Imbalanced	97.0	97.7	97.8	95.6	97.7	4.4	96.7
HE	Imbalanced	96.7	97.6	97.4	94.8	97.5	5.0	96.0
AGC	Imbalanced	97.2	98.0	97.8	95.6	97.9	4.4	97.0

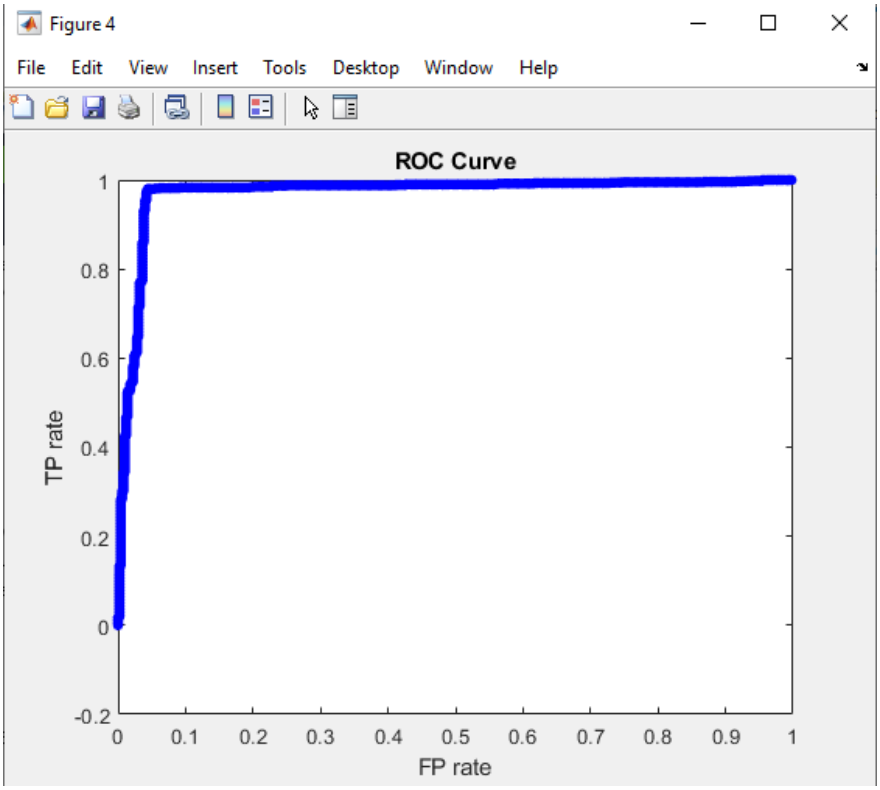
Table 9: Pre-processing effect on HH-SVM-HOG (%)

Likewise for the Full image model, the best HOG model has **Automatic Gamma Correction** and uses the **full imbalanced data set**.

The differences between the balanced and imbalanced data set for this SVM model is smaller than the full image equivalent. The HOG feature descriptor appears to extract enough information to make up for the under sampling performance cost associated with the smaller data set.

The effect of using AGC on this model is less noticeable than the full image equivalent. The gain in accuracy was 0.3% (96.9 $\rightarrow$ 97.2) and AUC improvement of 0.8% (96.2 $\rightarrow$ 97.0)

Below is the ROC for HH-SVM-HOG with AGC and full imbalanced data set:



Looking at the ROC it appears that HH-SVM-HOG with AGC and full imbalanced data set is great model for pedestrian classification with an AUC of 97.0%. Using the information obtained from this and the previous section of PCA tests will be only with AGC and full imbalanced data set to save time.

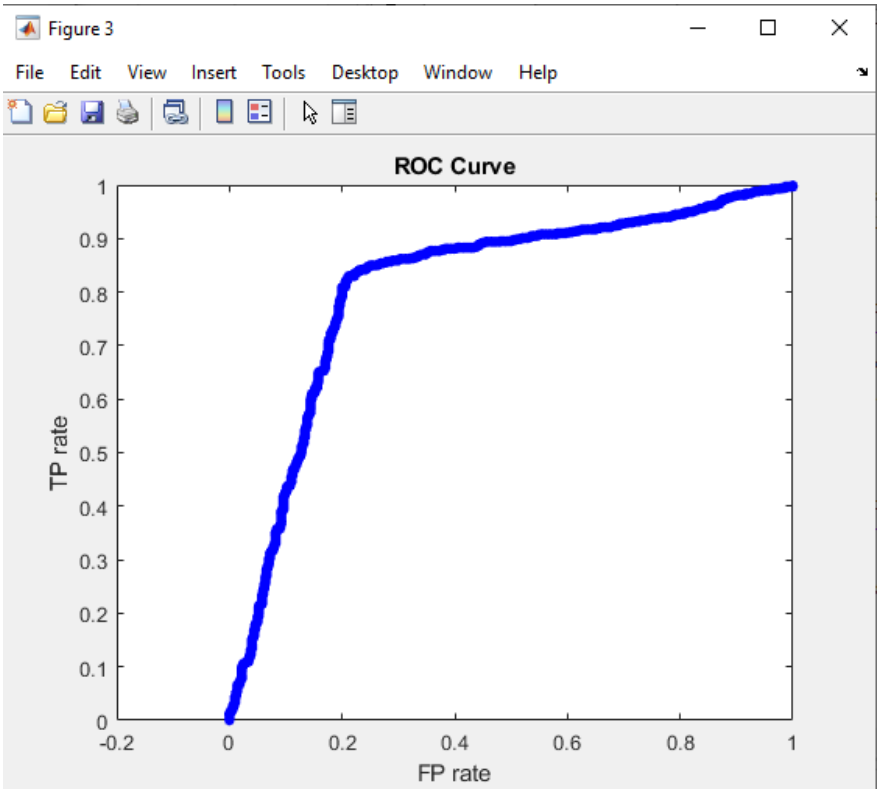
9.3 Half/Half SVM - PCA

ndim	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
default (500)	80.9	82.9	87.8	77.0	85.3	23.0	80.6
100	62.7	69.5	73.2	49.0	71.3	51.0	59.0
1000	81.6	83.0	88.7	78.8	85.7	21.2	81.2

Table 10: Number of dimensions effect on HH-SVM-PCA with AGC and full imbalanced data set (%)

HH-SVM-PCA with AGC and full imbalanced data set appears to have better performance compared to Full Image but worse than HOG. As ndim decreases the performance tends to decrease. This might be because PCA is removing useful information in the goal of reducing dimensions. It is likely more than 1000 dimensions is required for a good HH-SVM-PCA model. As HOG is appearing to out-perform PCA there is no reason to pursue the optimal ndim value.

Below is the ROC for HH-SVM-PCA with AGC and full imbalanced data set:



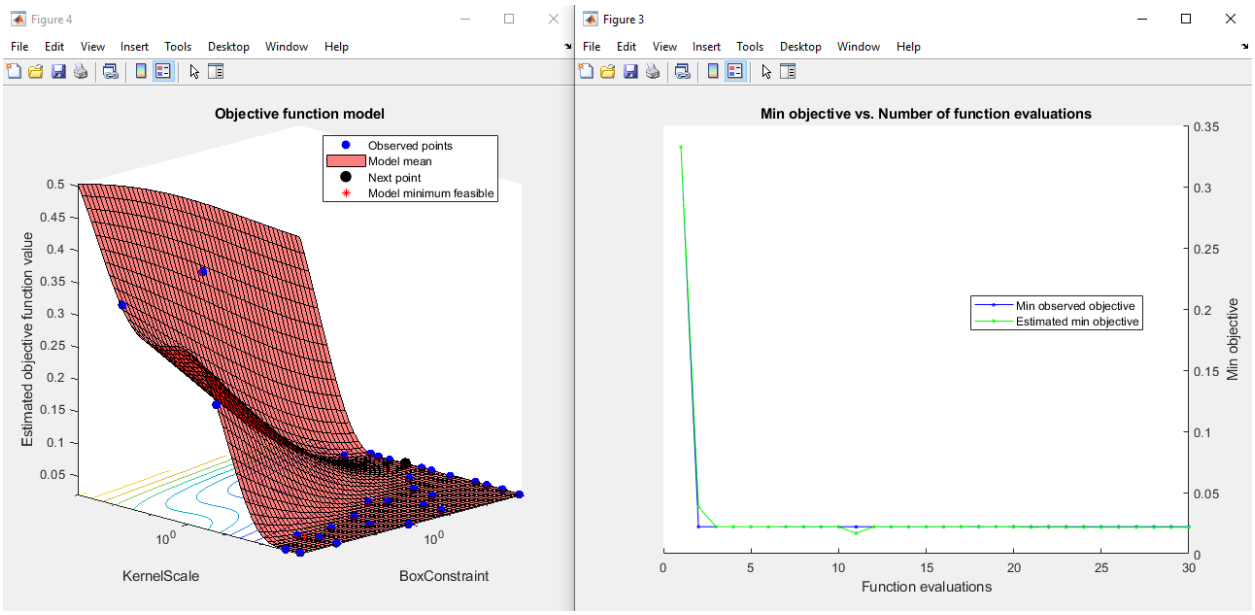
The AUC for PCA is similar to Full Image but much lower than HOG. It is unlikely that PCA is a good feature descriptor for the SVM model and pedestrian detection. Moving forward we will take the best HH-SVM-HOG model and perform cross validation in the aim to improve the fit of the model due to the imbalanced data set.

9.4 Cross-validated SVM - HOG with Hyper-parameter Optimisation

HH-SVM-HOG with Automatic Gamma Correction and full imbalanced data set seems to be the best performing model so far. Taking this model set-up and instead of using Half/Half for testing we can use cross validation and hyperparameter optimisation we can better fit the model for unknown data sets (such as being used in the detector). For Cross validation we will use the default set-up with various k-folds. For hyperparameter optimisation we will use the set-up discussed in the earlier SVM section.

Firstly, hyperparamter optimisation involves variation of different box constraints (cost) and kernel scale to find the best fitted model. For CV-SVM-HOG with AGC and full imbalanced data the results are as follows:

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	BoxConstraint	KernelScale
1	Best	0.33233	51.524	0.33233	0.33233	0.69776	293.9
2	Best	0.022	16.477	0.022	0.038412	0.0072031	0.0083882
3	Accept	0.33233	53.158	0.022	0.022031	0.0011075	69.597
4	Accept	0.029333	21.338	0.022	0.022017	203.11	13.052
5	Accept	0.022	16.192	0.022	0.021991	935.98	3.5
6	Accept	0.022	16.129	0.022	0.021977	3.4677	0.048419
7	Accept	0.022	16.24	0.022	0.021971	12.145	0.0018201
8	Accept	0.022	126.21	0.022	0.021961	0.0099065	0.0010043
9	Accept	0.022	16.346	0.022	0.021943	0.25731	0.0032855
10	Accept	0.022	16.691	0.022	0.021932	982.76	0.017527
11	Accept	0.218	52.309	0.022	0.01665	0.0010025	0.18139
12	Accept	0.022	16.767	0.022	0.021961	191.54	0.21179
13	Accept	0.022	128.24	0.022	0.021961	900.45	0.0010427
14	Accept	0.022	16.443	0.022	0.021913	48.64	0.012684
15	Accept	0.022	16.368	0.022	0.021918	893.22	0.077294
16	Accept	0.022	17.05	0.022	0.0219	0.0010006	0.0024254
17	Accept	0.022667	18.082	0.022	0.021895	998.39	12.12
18	Accept	0.022	17.446	0.022	0.021836	61.031	0.050878
19	Accept	0.022	16.567	0.022	0.021952	958.62	0.0031663
20	Accept	0.022	16.202	0.022	0.021827	0.010586	0.0031811
Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	BoxConstraint	KernelScale
21	Accept	0.022	16.252	0.022	0.021665	0.39307	0.01337
22	Accept	0.022	16.153	0.022	0.021692	975.45	0.48729
23	Accept	0.022	125.45	0.022	0.021675	0.87871	0.0010093
24	Accept	0.022	16.177	0.022	0.021659	947.37	7.0405
25	Accept	0.022	126.69	0.022	0.021661	0.0010409	0.0010148
26	Accept	0.022	16.385	0.022	0.021674	959.69	0.26347
27	Accept	0.022	16.644	0.022	0.021678	6.2494	0.026097
28	Accept	0.022	16.177	0.022	0.021635	0.049367	0.0070347
29	Accept	0.022	16.658	0.022	0.021628	976.73	0.0084521
30	Accept	0.022	16.512	0.022	0.021621	12.214	0.0054167



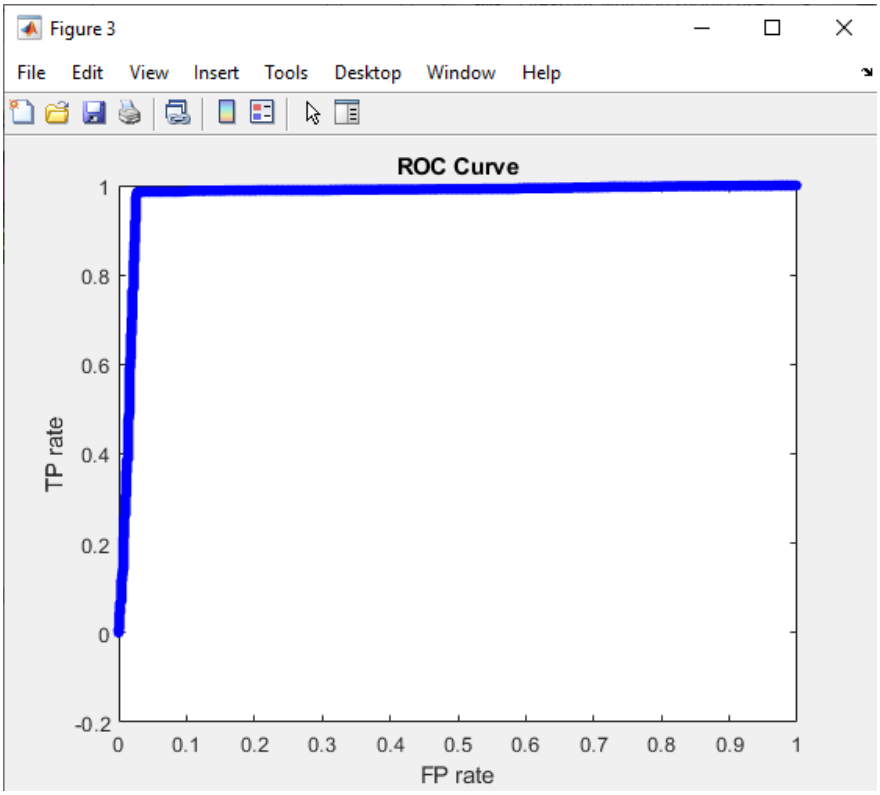

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 1074.7431 seconds.
Total objective function evaluation time: 1048.8708

Best observed feasible point:
    BoxConstraint    KernelScale
    _____________    ___________

      0.0072031       0.0083882

Observed objective function value = 0.022
Estimated objective function value = 0.021621
Function evaluation time = 16.4769

Best estimated feasible point (according to models):
    BoxConstraint    KernelScale
    _____________    ___________

      0.049367        0.0070347

Estimated objective function value = 0.021621
Estimated function evaluation time = 16.176
```

Above is the automatic 30 iteration process of varying the discussed hyperparameters to find the global minimum of values that produce the best fitted CV-SVM-HOG model. We will use the best observed feasible point values moving forward, therefore, for cross validation testing we will be using a **BoxConstraint** (cost) value of **0.0072031** and a **KernelScale** value of **0.0083882**.

k-fold	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
5	97.8	98.3	98.4	96.8	98.4	3.2	97.4
10	98.1	98.6	98.7	97.3	98.6	2.7	97.8
30	98.2	98.7	98.6	97.2	98.6	2.8	97.9

Table 11: Cross validation k-fold effect on CV-SVM-HOG with AGC and full imbalanced data set with cost of 0.0072031 and kernel scale of 0.0083882 (%)

The evaluation metrics for CV-SVM-HOG are consistent for different k-values this means the model is well fitted. Below is the ROC for CV-SVM-HOG:



The AUC is 97.8% which is the highest achieved so far and shows little room for improvement. CV-SVM-HOG may be our best model. In the next section we will compare with Random Forest models.

10 Model Evaluation - RF

10.1 Half/Half RF - Full Image

Method	Data Set	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
-	Balanced	85.1	94.1	71.1	80.4	81.0	19.6	78.7
ALS	Balanced	83.8	93.8	69.2	78.6	79.6	21.4	76.9
HE	Balanced	81.5	96.2	65.5	74.1	77.9	25.9	74.7
AGC	Balanced	86.5	96.2	72.7	81.5	82.8	18.5	81.4
-	Imbalanced	92.4	96.8	92.2	83.6	94.4	16.4	89.9
ALS	Imbalanced	90.7	95.7	90.9	80.8	93.2	19.2	88.0
HE	Imbalanced	89.2	96.5	88.4	74.6	92.3	25.4	85.8
AGC	Imbalanced	92.0	96.6	91.8	82.8	94.2	17.2	89.5

Table 12: Pre-processing on HH-RF-Full Image with 128 trees (%)

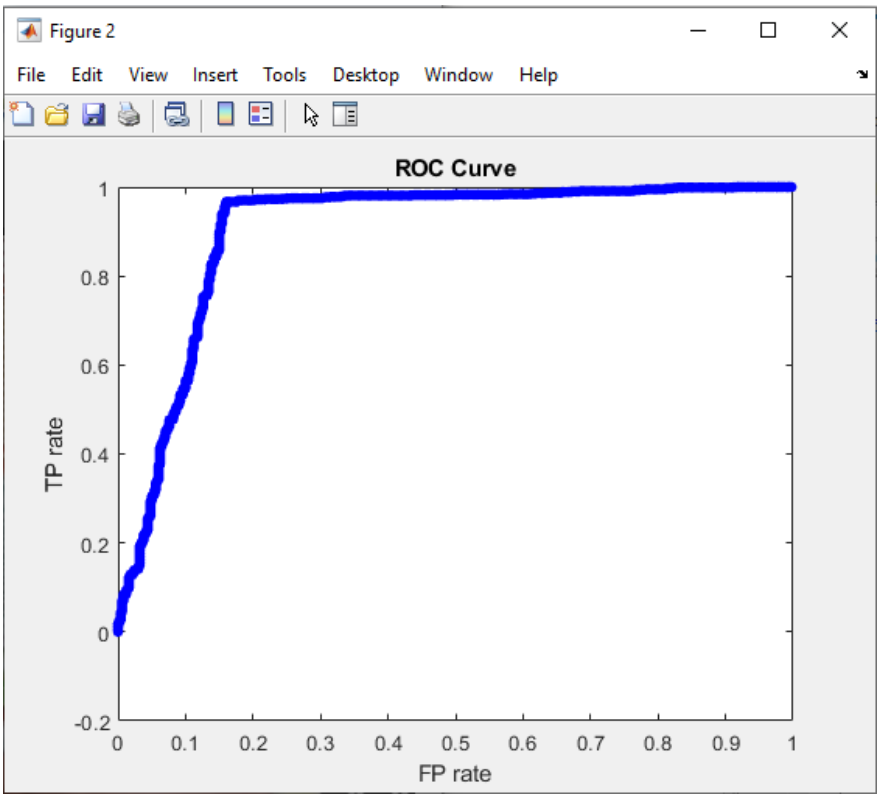
The HH-RF-Full Image model without pre-processing and the full imbalanced data set appears to have performed the best. However, pre-processing nor using the smaller balanced data set produced very similar results overall. Random forest appears to have performed well under most conditions.

ntrees	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
64	92.0	96.6	91.8	82.8	94.2	17.2	89.6
128	92.4	96.8	92.2	83.6	94.4	16.4	89.9
256	92.5	96.7	92.4	84.0	94.5	16.0	90.2

Table 13: Number of trees effect on HH-RF-Full Image model with no pre-processing and the full imbalanced data set (%)

Changing the number of trees does not seem to alter the performance much. It appears a higher number of trees at the value of 256 produces the best model.

Below is the ROC for the best model:



With an AUC of 90.2% the model appears to perform well, however, it performs worse than the best SVM model from the previous section. Perhaps a different feature descriptor could help.

10.2 Half/Half RF - HOG

Method	Data Set	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
-	Balanced	89.1	99.4	75.8	83.8	86.0	16.2	85.4
ALS	Balanced	89.5	98.2	77.0	85.0	86.3	15.0	85.8
HE	Balanced	92.5	99.4	82.2	88.9	90.0	11.1	89.8
AGC	Balanced	87.8	98.8	73.9	82.1	84.6	17.9	83.5
-	Imbalanced	94.0	98.1	93.3	85.8	95.6	14.2	91.7
ALS	Imbalanced	94.3	97.2	94.5	88.6	95.8	11.4	92.4
HE	Imbalanced	95.6	99.1	94.6	88.6	96.8	11.4	93.3
AGC	Imbalanced	93.4	97.3	93.1	85.6	95.2	14.4	91.6

Table 14: Pre-processing on HH-RF-HOG with 128 trees (%)

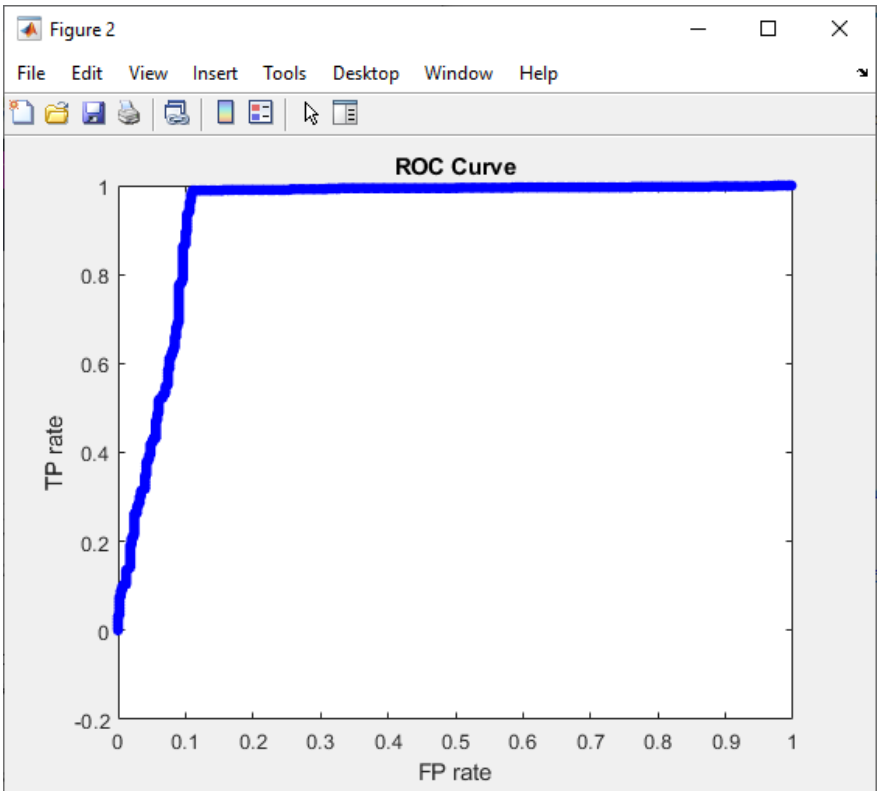
The HH-RF-HOG model with Histogram equalisation and the full imbalanced data set appears to have performed the best.

ntrees	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
64	95.3	99.0	94.3	88.0	96.6	12.0	93.0
128	95.6	99.1	94.6	88.6	96.8	11.4	93.3
256	95.6	98.9	94.7	89.0	96.8	11.0	93.5

Table 15: Number of trees effect on HH-RF-HOG model with HE and the full imbalanced data set (%)

Changing the number of trees does not seem to alter the performance much. It appears a higher number of 256 produces the best model.

For PCA we will just use the full imbalanced data set because the extra observations appear to help the performance of the models more than the smaller balanced set. Below is the ROC for the best HOG model:



The HOG RF model appears to achieve a better AUC as the full image model (HOG: 93.5% Full Image: 90.2%), however it is not as high as HOG SVM (97.9%).

10.3 Half/Half RF - PCA

Method	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
-	80.5	99.9	77.4	41.6	87.2	58.4	69.7
ALS	81.1	100.0	77.9	43.3	87.6	56.8	70.4
HE	80.1	99.2	77.4	42.0	86.9	58.0	69.2
AGC	82.7	99.6	79.6	48.8	88.5	51.2	73.4

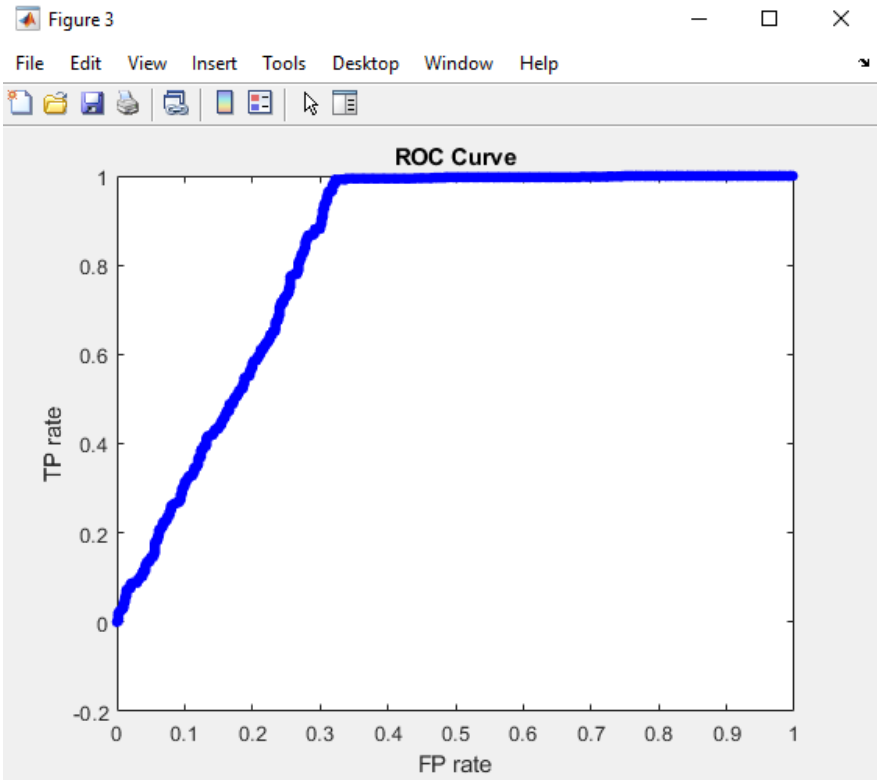
Table 16: Pre-processing on HH-RF-PCA (ndims = 500) with full imbalanced data set (ntrees = 256) (%)

The best performing HH-RF-PCA was with the full imbalanced data set and automatic gamma correction pre-processing.

ndim	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
default (500)	82.7	99.6	79.6	48.8	88.5	51.2	73.4
100	88.7	99.2	86.0	67.8	92.2	32.2	82.8
1000	80.7	99.9	77.6	42.2	87.3	57.8	70.2

Table 17: Number of dimensions effect on HH-RF-PCA with AGC and full imbalanced data set (ntrees = 256)(%)

It appears that HH-RF-PCA performs best with less dimensions achieved up to 88.7% accuracy. Below is the ROC for the best model:



The AUC for this model is 82.8% which is over 10% less than the best SVM Model.

11 Choice of Model for Pedestrian Detection

The best performing model was CV-SVM-HOG with Automatic Gamma Correction and with the full imbalanced data set. This model has the highest Accuracy and AUC out of the other models. It has also been cross-validated and has fine tuned hyper paramaters to reduce the effect of over fitting on the training set. Therefore, we expect this model to perform best in pedestrian detection. This model was expected be the best as HOG is a feature descriptor specifically designed pedestrian detection and SVM appears to be the most robust classifier for the problem. The SVM model implementation appears to be fastest out of them all during testing meaning for real time detection it could be the best model.

Model	Accuracy	Recall	Precision	Specificity	F-measure	False alarm rate	AUC
NN	84.3	73.9	78.6	89.7	76.2	10.3	69.7
KNN	84.6	87.6	89.2	78.8	88.3	21.2	82.9
SVM	98.2	98.7	98.6	97.2	98.6	2.8	97.9
RF	88.7	99.2	86.0	67.8	92.2	32.2	82.8

Table 18: Summary of the best model for each classifier type (%)

Going forward the config.txt file will be configured for CV-SVM-HOG with AGC and full imbalanced data set. All detector testing and analysis will be with that model.

12 Detectors

12.1 Initial Sliding Window Detector

A sliding window detector is a great way to analyse an image for certain objects(in our case people). It allows the image to be split up into overlapping parts which are passed individually to the model for classification. The main benefit of this is that from the results, we can begin to identify which locations of the image contain a person.

We implemented this function as follows:

```
function [b_boxes] = sliding_window2(image,scale,model,config)
    row = 1;
    col = 1;
    [maxCol, maxRow] = size(image);

    %divisor determines how much windows overlap.
    %EG: 1 = no overlap
    col_sep = scale(2)/7;
    row_sep = scale(1)/7;

    windowMax = 1;
    for y = col:col_sep:maxCol-scale(2)
        for x = row:row_sep:maxRow-scale(1)
            windowMax = windowMax+1;
        end
    end
```

The function takes in an **image** (structured as a 2d matrix), **scale**(e.g: [160,96], **model** and **config**. The scale represents the dimensions of the crop/window that we want to separate the full image into. An important requirement of sliding window detection is that the windows overlap with each other. This overlapping avoids us losing results due to the person being split apart by the windows.

In order to implement this overlapping we first decided on separation values for the columns and rows. This was calculated by using the respective scale value and dividing it by an integer. The bigger the integer means a greater overlap. (This can be seen in the above function snippet)

The next step was to calculate how many windows we can actually fit on the image given the scale and desperation values. This was calculated using two for loops iterating through all the columns and rows and counting how many windows we will have. (Seen as the final code segment from the above function snippet). This max window value is used to initialize our results and bounding box variables as they will be of length:max window value.

```
        for y = col:col_sep:maxCol-scale(2)
            for x = row:row_sep:maxRow-scale(1)
                po = [x, y, scale(1), scale(2)];
                img = crop_img(po, image);
                img = imresize(img,[160,96]);
                cd ../
                %What feature destriptor will we use: HOG or PCA
                cd feature-descriptor
                if strcmp(config.featureDescriptor,"HOG")
                    feature_vector = hog_feature_vector(img);
                end
```

As seen in the above code we start our actual sliding window detection by iterating using two for loops which go from **1-maxcol/row - the scale** and iterate by the separation values previously calculated to insure overlap.

For each iteration we store as a variable the **x and y coordinates along with each scale value representing width and height**. In matlab this data structure is used to represent a box...which in turn can be used to crop from a full image. We created an auxiliary function which takes in this box data structure along with a full image and returns the cropped image. This image was then resized to [160,96] using the built in **imresize** function. We needed the crop to be these dimensions so that it would be compatible with our model classifier.

In order for the detector to know which feature descriptor/classifier to use, the function checks what string values are stored inside the config passed in variable. We gave the detector the ability to use **HOG**,**PCA**,**Half-**

```
                if strcmp(config.classifier,"SVM")
                    cd classifier/svm
                    if(strcmp(config.dd,"HH"))

                    results(windowNumber) = SVMTesting(feature_vector,r

                    end
```

Half,**Cross-Validation**,**SVM**, and **KNN**.

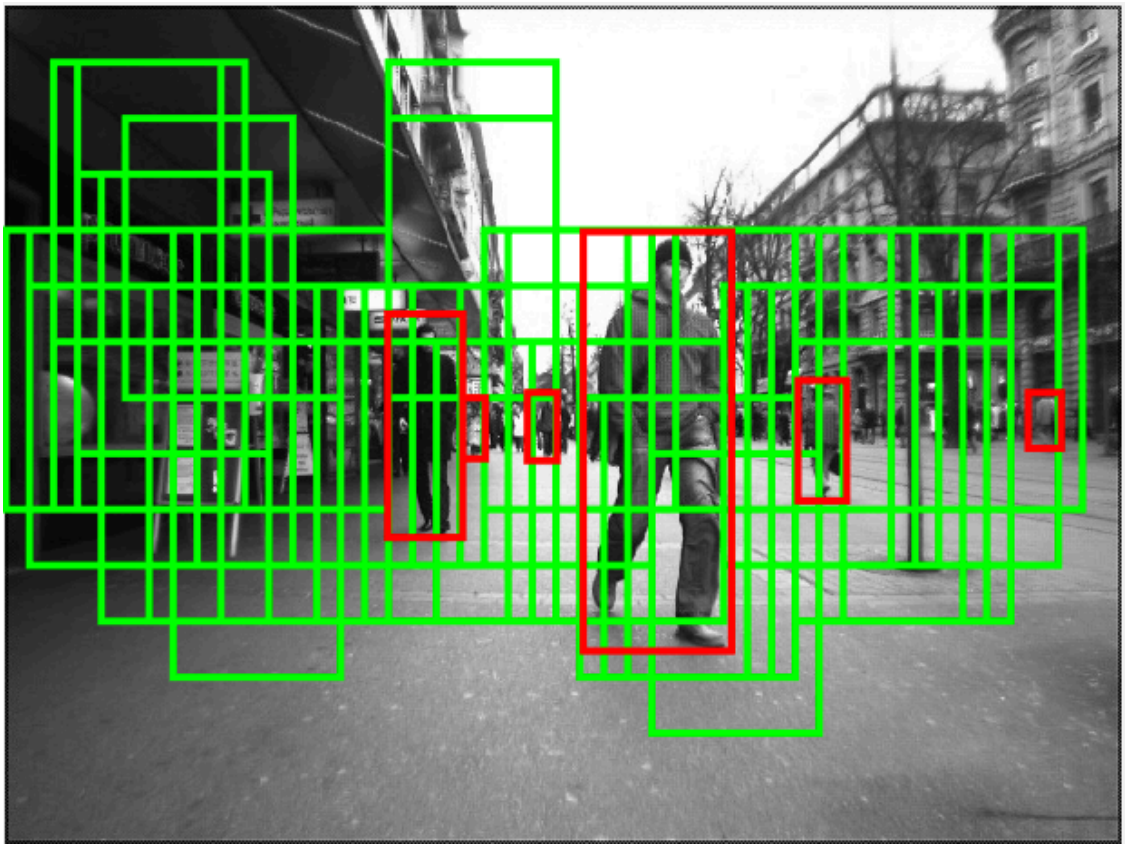
Here is an example from the function if we are wanting to use a **SVM classifier with half and half data division**. We can see that the result is stored into a variable called **result** at an index of 'windowNumber' which iterates after every box.

```
                end
                cd sliding-window
                boundingBox(windowNumber, 1) = x;
                boundingBox(windowNumber, 2) = y;
                boundingBox(windowNumber, 3) = scale(1);
                boundingBox(windowNumber, 4) = scale(2);
                windowNumber = windowNumber+1;
            end
        end
        b_boxes = boundingBox(results == 1, :);
    end
```

Finally we store the windows **coordinates, width and height** in variable matrix called 'boundingBox'. After iterating through the whole image the function **returns all bounding boxes which gave a positive result from the classifier**. These boxes can then be used for Non-Maxima Suppression.

```
function [b_boxes] = apply_swd(images,img_num,n
    % - Image vector to apply HOG
    img_index = img_num;
    Im_vector = images(img_index,:);
    % - HOG implementation requires 2D images
    Im_reshape = reshape(Im_vector,num_row,num_

    [b_boxes] = sliding_window2(Im_reshape,scal
end
```

The sliding window detection function is called from another function called 'apply swd'. This functions main purpose is to **restructure the image vector into a 2d matrix**. It also aids us when implementing multiple scales.



From the image above we can see our sliding window detectors bounding boxes (with scale [96,160]) in green and the ground-truth boxes in red. Each green bounding box shown had a positive result from the model. As we can see there is a large amount of overlapping however it's a good initial first step as there is a lot of coverage over the people and not too many boxes are extremely far off.

12.2 Multi-Scaling

Having a sliding window detector is a good start, however, for it to reach it maximal potential we will want to give it the option to classify the image using multiple window scales. This will allow the detector to pick up people at different ranges. For example a smaller scale will pick up a person in the distance while a large scale will detect a person who is larger and at the foreground of the image. Having multiple scales also allows us (once non-maxima suppression is implemented) to prioritise boxes closer to the same dimensions as the person making the resulting boxes more accurate.

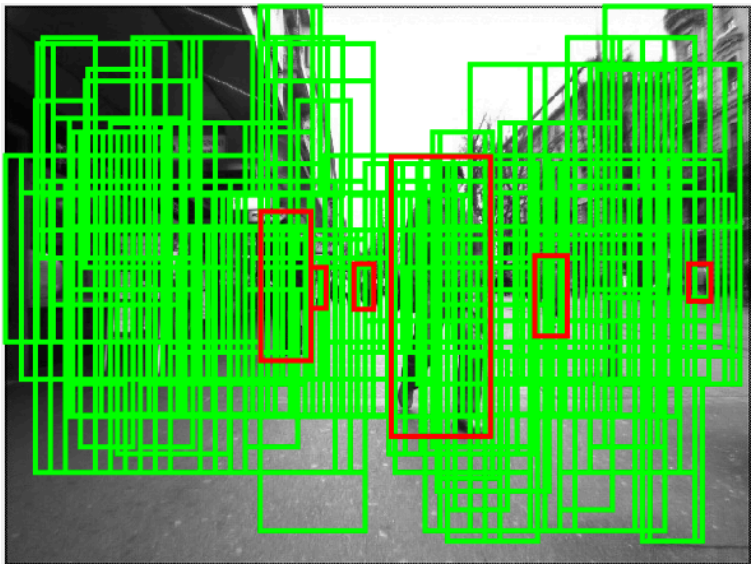
```
%Scales to use
scales = [[400,600];[70,150];[100,300]];
```

In order to achieve this, we first store the scales we want to use within this variable 'scales'.

Like shown in **Initial Sliding Window Detector (Section 8.1)**, We had a function called 'apply swd' that applies the detector onto an image.

```
for i=1:img_num
    cd sliding-window
    disp("Image number: "+i);
    for j=1:length(scales)
        disp("Processing Scale: ["+scales(j,1)+","+scales(j,2)+"]");
        if j==1
            b_boxes = apply_swd(pp_pedestrian_images,i,num_row,num_col,modelSVM,scales(j,:),config);
        else
            b_boxes = [b_boxes; apply_swd(pp_pedestrian_images,img_num,num_row,num_col,modelSVM,scales(j,:),config
        end
```

We can then achieve a multiple scaling sliding window detector by iterating through each scale for each image. All we have to do is ensure that if its the first scale we save all the bounding boxes to the variable 'b boxes'. and for every other scale after that we just append the bounding boxes onto the variable. This means that when all scales have been checked we will have one variable which stores all the bounding boxes from all the selected scales.



Here are the results from a multi-scaling sliding window with window scales of: [96,160],[90,250],[45,135]. As we can see, we are definitely picking up the ground truth people however with the amount of overlapping it is really hard to make out which boxes are accurate and which are not. All this overlapping is noise, therefore we will be looking into reducing this by using Non-Maxima Suppression.

13 Non-Maxima Suppression

13.1 Purpose

As seen within **8.2: Multi-Scaling**, There is a lot of noise generated with all the different sized bounding boxes overlapping each other. Due to this noise it's near impossible to make out wbhich people are being detected by what box and where they are. A process called **Non-Maxima Suppression** will help with reducing noise by culling a portion of the bounding boxes that overlap each other over a certain threshold.

13.2 How do we decide which boxes to use

The first step is to **first identify which boxes overlap with one another over a certain threshold**. This threshold represents the area of the overlap between the two boxes.

We can calculate the intersection area of two bounding boxes by using the matlab function **rectint**.

```
overlapping_boxes = [];
for i=1:size(b_boxes,1)
    cntr = 1;
    for j=1:size(b_boxes,1)
        if j ~= i
            overlap = rectint(b_boxes(i,:),b_boxes(j,:));
            if overlap > threshold
                overlapping_boxes(i,cntr) = j;
                cntr = cntr + 1;
            end
        end
    end
end
```

This code snippet from our Non-Maxima Suppression function shows how we chose to store these boxes. As you can see we are looping through our bounding boxes twice, calculating the overlap between them and then storing the second boxes index as a column on the first box indexes row. This basically will leave us with a matrix where every row number represents each boxes index and every column value is an index of a box that overlaps with it. This will make it easier for us when we come to calculating which boxes to delete.

13.3 Confidence Values

Confidence Values are what we will compare between two overlapping boxes to decide which one to remove. We will want to remove the lower of the two.

Since we have the luxury of having the 'ground truth' bounding boxes for each image(The desired output for our detector), we decided to use the overlap between our box and the ground truth boxes as our confidence value.

```
function [best_cv] = best_confidence(box_in,img_boxes)
    best_cv = 0;
    for i=1:size(img_boxes,1)
        img_box = img_boxes(i,:);
        box_intersect_area = rectint(box_in,img_box);
        union_area = (box_in(3)*box_in(4))+(img_box(3)*img_box(4))-box_intersect_area;
        value = box_intersect_area/union_area;

        if value > best_cv
            best_cv = value;
        end
    end
end
```

Here is our way of calculating the 'best' confidence value for a given box. All this function is doing is comparing the overlap of the box to each images ground truth boxes and returning the greatest value.

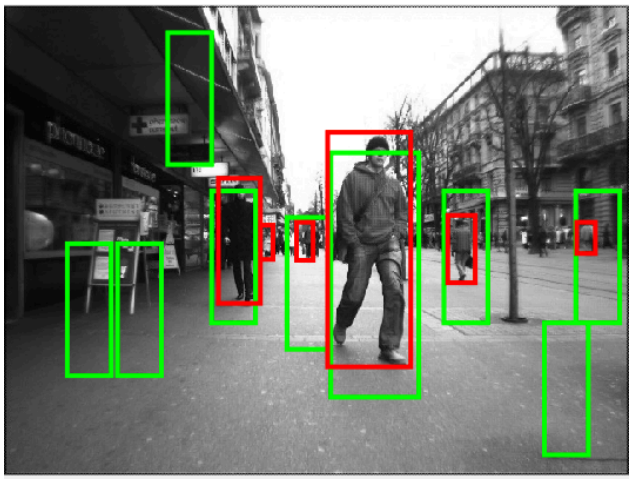
13.4 Deciding Which Boxes To Remove

Now that we know which boxes overlap over the threshold and have a way of calculating confidence values, we can now decide which boxes to remove:

```
deleted = [];
for i=1:size(overlapping_boxes,1)
    box_i = b_boxes(i,:);
    cv_i = best_confidence(box_i,img_boxes_rs);
    area_i = box_i(3)*box_i(4);
    for j=1:size(overlapping_boxes,2)
        temp_index = overlapping_boxes(i,j);
        if temp_index ~= 0 && ~(ismember(i,deleted)) && ~(ismember(temp_index,deleted))
            box_temp = b_boxes(temp_index,:);
            cv_temp = best_confidence(box_temp,img_boxes_rs);
            area_temp = box_temp(3)*box_temp(4);
            %Do we have to delete box_i or not
            if cv_temp > cv_i
                %Delete box_i
                deleted = [deleted;i];
            else
                %If both have 0 cf, delete box_i if box is smaller
                if cv_i == cv_temp
                    if area_i <= area_temp
                        deleted = [deleted;i];
                    end
                end
            end
        end
    end
```

This is our initial singular approach to deciding which boxes to delete. Remembering back to how we stored the overlapping boxes where each row is a box index and all the columns contain the index of boxes that overlap with it. Therefore as we are iterating we are storing the bounding box at index 'i', checking to see if it overlaps with any box with a greater confidence value than it and deleting it if that's the case. We also added an extra case which checks if both confidence values are zero and deleting the i'th box if its smaller than or equal to the one we are comparing to.

The reasoning behind keeping the bigger box when both overlapping boxes have zero confidence is that the bigger box is more likely to contain a person and will ultimately reduce the number of final boxes.



The above image was processed with the same scales as the result generated in **Multi-Scaling** but with our basic Non-maxima suppression implementation as explained in the previous paragraphs. We also used a threshold value of 0.5. As we can see there are basically no overlapping boxes anymore and we can clearly see what each remaining box is picking up. From this image we can see that our model picked up all bar one of the people. Due to our calculation for confidence value we can see that the detector is prioritising boxes that are closer to the area of the actual person.

13.5 Improvements

This initial Non-Maxima Suppression function works well however we were interested in ways we could enhance it's logic. The main thing we decided to look into is the idea of classifying any box that has a confidence value of over a given threshold as 'correct'. Our idea was to look for and store the most 'correct' box for each ground-truth box in the image. We could then ensure that these boxes are not deleted during NMS deletion.

```
function [correct] = correct_overlaps(b_boxes,img_boxes,nms_gt)
    %Store all boxes which cross overlap threshold with ground truth boxes (img_boxes)
    correct = [];
    for i=1:size(img_boxes,1)
        %Check overlaps for each box
        cntr = 1;
        for j=1:size(b_boxes,1)
            b_box = b_boxes(j,:);
            img_box = img_boxes(i,:);
            box_intersect_area = rectint(b_box,img_box);
            union_area = (b_box(3)*b_box(4))+(img_box(3)*img_box(4))-box_intersect_area;
            overlap = box_intersect_area/union_area;

            if overlap >= nms_gt
                correct(i,cntr) = j;
                cntr = cntr + 1;
            end
        end
    end
end
```

We implemented this above function which takes in all the bounding boxes from our detector, the ground truth boxes and a threshold. Our function records all boxes which have a confidence value over the threshold for each ground-truth box. We then return this data within a similar data structure to our **overlapping boxes** where each row number corresponds to the index of the ground-truth box.

The next step we took was to pick the highest confidence valued box for each ground-truth.

```
%i = index of each img
for i=1:size(correct,1)
    best_index = 0;
    best_cv = 0;
    best_area = 0;
    %j will iterate through all indexes of bbox recorded as correct for
    %i
    for j=1:size(correct,2)
        if j == 1 && correct(i,j) ~= 0 && ~(ismember(correct(i,j),deleted))
            best_index = correct(i,j);
            best_cv = crop_confidence(b_boxes(best_index,:),img_boxes_rs(i,:));
            best_area = (b_boxes(best_index,3)) * (b_boxes(best_index,4));
        else
            if correct(i,j) ~= 0 &&  ~(ismember(correct(i,j),deleted)) &&  ~(ismember(correct(i,j),kept_correct))
                temp_index = correct(i,j);
                temp_cv = crop_confidence(b_boxes(temp_index,:),img_boxes_rs(i,:));
                temp_area = (b_boxes(temp_index,3)) * (b_boxes(temp_index,4));
                if temp_cv > best_cv
                    best_index = temp_index;
                    best_cv = temp_cv;
                    best_area = temp_area;
                else
                    if temp_cv == best_cv
                        if temp_area < best_area
                            best_index = temp_index;
                            best_cv = temp_cv;
                            best_area = temp_area;
                        end
                    end
                end
            end
        end
    end
    %Add best_index to kept_correct
    kept_correct = [kept_correct; best_index];
end
```

This is the code snippet from the new NMS function which iterates through all the 'correct' boxes and keeps track of which has the highest confidence value for each ground-truth. It then stores the best box for the i'th ground-

truth box in the variable **kept correct**.

Now that we have a record of the most 'correct' boxes we can give them priority over the slightly less 'correct' ones.

Next we just repeat the same removal process as shown in **12.4** where we decide what all boxes to delete.

All that's left to do is a final check which ensures our 'correct' boxes remain and any boxes that overlap with them over the **nms threshold** are deleted.

```
%Any remaining overlap with our 'correct' boxes
add_these = [];
for i=1:size(overlapping_boxes,1)
    if ~(ismember(i,deleted)) && ~(ismember(i,kept_correct))
        for j=1:size(overlapping_boxes,2)
            if (ismember(overlapping_boxes(i,j),kept_correct))
                %delete i
                deleted = [deleted;i];
                if ismember(overlapping_boxes(i,j),deleted)
                    %Add j
                    add_these = [add_these;overlapping_boxes(i,j)];
                end
            end
        end
    end
end
```

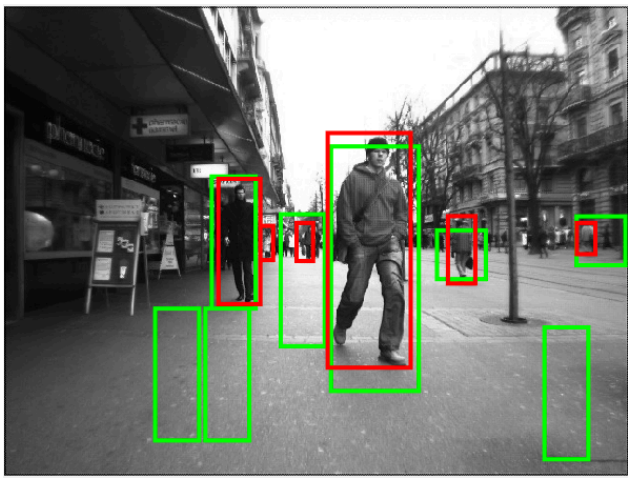
The above snippet shows how we do this. We iterate through each box, if it has not been deleted and is not a 'correct' box but overlaps (over threshold) with a box that is 'correct' we delete it. We also perform a check to see if the 'correct' box was accidentally deleted, it it was we add it to a temp list which will be re-included to the final boxes.

The code snippet which creates outboxes is very slightly different to that one in the basic NMS implementation:

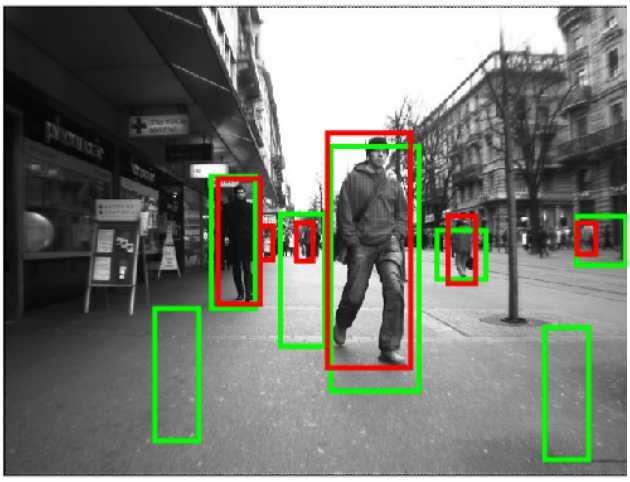
```
%Create output box matrix
cntr = 1;
for i=1:size(b_boxes,1)
    if ~(ismember(i,deleted)) || (ismember(i,add_these))
        nms_boxes(cntr,:) = b_boxes(i,:);
        cntr = cntr + 1;
    end
end
```

As you can see above we just add an **OR** check which basically allows us to add any 'correct' boxes that were accidentally deleted.

This improved Non-Maxima Suppression will only yield better better results than our basic NMS function IF we use scales that will have an overlap ratio greater than our given **Ground-Truth Threshold**(Set at 0.60). For example if we use the scales [[50,50];[96,160];[90,250];[45,135]] with basic NMS we get:

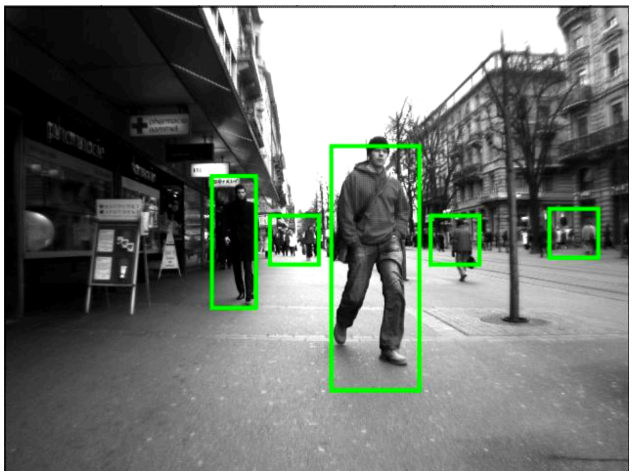


And with the improvement mentioned implemented with the same scales, model and a ground-truth threshold of 0.6 we get:



Both functions return almost identical results with the improved version containing one less false positive. Both missed the smallest pedestrian, the reason for this is because a box that will pick it up overlaps with the really strong box to its left it will be deleted. Our improved also misses it because it only save the box if the overlap ratio is greater than 0.6 and currently we aren't using a scale small enough to have a ratio that high.

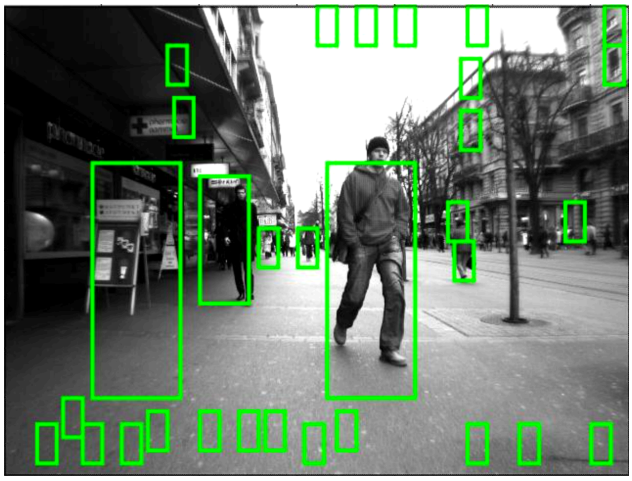
To prove that we will start picking that small pedestrian up with our improved function we will lower our ground-truth threshold to 0.2



By reducing the ground-truth threshold we can see what happens when the function starts identifying some of our boxes as 'correct'. You will notice that for each pedestrian a box of very similar size is used and no boxes are accidentally deleted.

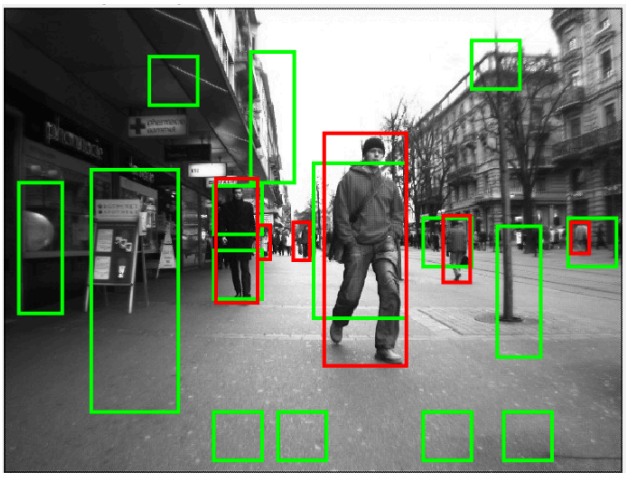
We decided to test different scales to use. Our idea was to go for a **small,medium, large** approach. We used the scales [20,40];[50,130];[90,240] initially. This allows us to pick up a decent variety of sized pedestrians without having to process too many boxes(which would happen if we added more scales).

We then needed to decide on what threshold to use for our priority feature. As we are opting to use less scales, we wont be able to have the threshold as high as we would like (approx 0.6) due to it being unlikely the scales ratio would be above that on average. Therefore we experimented with lowering it and observing the results.



This image was generated with a threshold of 0.4, as you can see we are picking up the pedestrians well but with the smallest box size we get quite a few false positives.

We then tried a threshold of 0.3 with slightly adjusted scales of [50,50];[96,160];[90,250];[45,135] to make our smallest a bit bigger to help reduce false positives. Here is the result:



As you can see we get slightly better results compared with 0.4, however we are still getting a big ratio of false positives as well as two boxes detecting the same pedestrian.

A 0.2 threshold was already explored earlier and it still contains the best results from all our tests.

13.6 Evaluation Methods

In order to evaluate our detector we first had to extract values from the process

```
      num_correct = 0;
    for i=1:size(correct,1)
        present = false;
        for j=1:size(correct,2)
            if correct(i,j) ~= 0
                present = true;
            end
        end
        if present
            num_correct = num_correct + 1;
        end
    end

     %How many people are in the image
    total_people = size(img_boxes_rs,1);

    %Use nms_boxes as total amount of boxes since b_boxes will have
    %multiple scales picking up same false positive
    total_boxes = size(nms_boxes,1);

    %How many people did our detector miss
    missed = total_people - size(correct,1);

    %Accuracy against false positives
    accuracy = (num_correct/total_boxes) * 100;
```

We extracted an accuracy(ratio of correct boxes to all output boxes) and number of people NOT registered as 'correct'.

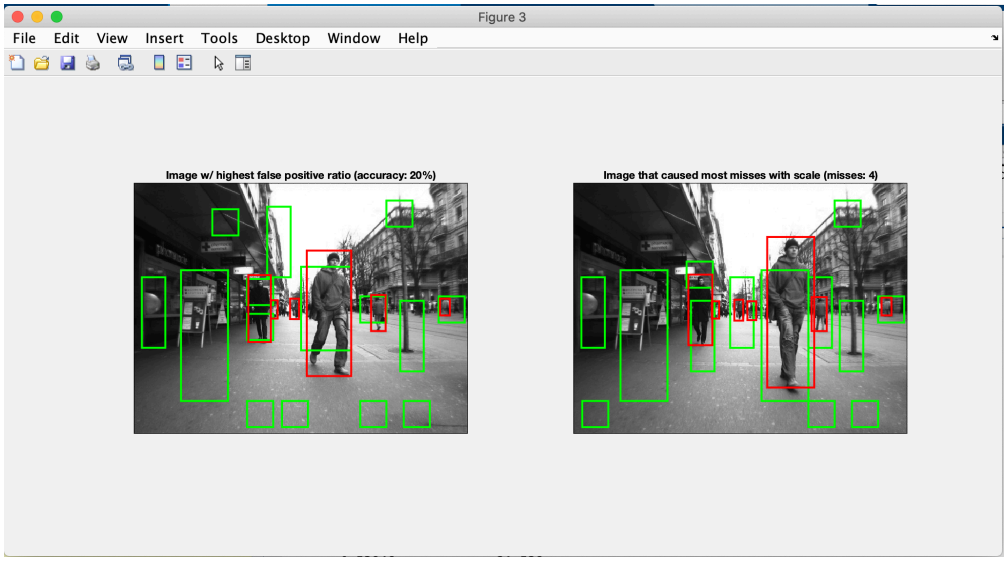
Using these statistics we can keep track of the average accuracy and average number missed throughout all images ran. We also keep track of the image with the lowest accuracy along with the image with the most people classified as missed.

These values are passed into an evaluation function which displays all the information onto the console like this:

```
--SWD EVALUATION: PARAMS--

--Scales used for window--
1: [20,40]
2: [50,130]
3: [90,240]

    num_scales    nms_overlap    nms_groundtruth
    __________    ___________    _______________

        3             0.5             0.4

--SWD EVALUATION: RESULTS--
Misses: how many times detector doesn't pick up person within a certain ratio
Accuracy: ratio of positive pickups to false positive

    average_misses    average_accuracy
    ______________    ________________

       0.15385            70.565
```

The function also shows a subplot figure showing the least accurate and image with the most missed from

the set processed:



13.7 Final Thoughts

From the options we tested we found the best results with a 0.5 overlap threshold, 0.2 ground-truth, multi-scaling detector using scales: **[[50,50];[96,160];[90,250];[45,135]]** for the improved NMS function and **[[20,40],[50,130],[90,240]]** for the basic implementation. We experimented with higher ground-truth thresholds as shown but found that we would have to use a BIGGER variety of scales which would be unrealistic to process given the amount of images we want to go through.

We also found that really small scales produced a **larger amount of false positives** and were generally more counter-productive due to this and the exponentially longer time it took to process them.

For our final project we ended up with **two separate NMS functions**. The first is a basic one which doesn't strive for anything too fancy...it just compares confidence values and deletes. The second tries to do a bit more with ground-truth boxes...prioritising the most correct for each pedestrian and making sure they survive. Overall both functions produce fairly accurate final results. **An output video for each function will be included in the GitLab repository for analysis**.

14 Output Video

Our final goal apart from evaluation methods is to create a video which showcases all the images (frames) from the pedestrian dataset including the remaining bounding boxes from non-maxima suppression.

```
%% Pre Processed (What computer uses)
% create the video writer
writerObj = VideoWriter('swd_results_pp');
writerObj.FrameRate = 6;

for i=1:length(swd_boxes)
    img_index = i;
    %Extract bounding boxes from cell
    img_boxes = swd_boxes{1,img_index};

    %Create results image
    figure("Visible",false);
    imshow(mat2gray(reshape(pp_imgs(i,:),num_row,num_col)))
     hold on
     axis on
     for j=1:size(img_boxes,1)
         bounding_box = img_boxes(j,:);
         rectangle('Position',[bounding_box(1,1),bounding_box(1,2),bounding_box(1,3),bounding_box(1,4)],'EdgeColor', 'g','LineWidth',2)
         daspect([1,1,1]);
     end

     temp_frames(i) = getframe(gcf);
end
```

As seen in the above snippet we first have to create a **VideoWriter** object which will allow us to create a .avi file with a set framerate. After that we can iterate through the bounding boxes and images in order to create invisible figures. We can then use the function **getframe(gcf)** which will store the current figure as a frame. Once our loop terminates we will have a variable **temp frames** which will store all the frames that needed added to the video.

```
open(writerObj);
 for i=1:length(temp_frames)
     % convert the image to a frame
     frame = temp_frames(i) ;
     writeVideo(writerObj, frame);
 end

 close(writerObj);
```

In order to add these frames to the video we first have to open the **VideoWriter** object and then iterate through, adding each frame individually to it. Once all frames have been added we just close the **VideoWriter** object.

The output .avi video file is then **saved to the same file location** as our **Create Video** function.

Our Create Video function repeats this process so that we create two videos, one which shows the **default images w/ boxes** and another that shows the **pre-processed images w/ boxes**. Having access to both allows us to quickly compare between the two if we observe weird results on a certain frame.

15 Possible Future Improvements

If given more time on the project we would of definitely explored other ways to improve our pedestrian detection. In particular we would have experimented with: **Video Tracking , Custom Feature-Extractor and Part Detection**. As we already have the video output including tracking would be an obvious next step and we believe the Part Detection and Feature-Extractor would further improve our accuracy.