**CSC4067 - Advanced Intelligent Information Systems  
Pedestrian Detection**

**David Jack - 40057705  
Timothy O'Neill - 40062164  
Nathan Sullivan – 40059116**

**1. Introduction and Background**

The goal for this project is to implement a pedestrian detector in MatLab using various machine learning algorithms. The general process for this is to take a set of images which we know either contain or do not contain people, extract features from them using an algorithm like HOG or PCA and create a model from those features which other testing images can then be compared against to predict whether or not the image contains a person or not.

There are two main parts to our pedestrian detection; the feature extractor and the classifier. The role of the feature extractor is to reduce the amount of resources required to describe a dataset and allow the classifier to work on a more generalized set of data by extracting relevant information from the data.

The classifier is the part of the system which predicts whether or not a testing image matches one group or the other. In our case that is predicting whether or not an image contains a person. After being given a set of training data to learn from, the classifier will have a split between people and non-people and then when a testing image is given to it, it will extract the features from the testing image and see where it lies within the model.

An issue to keep in mind during feature extraction and classification is that of overfitting[1]. Overfitting can occur when the model is trained to too specific a feature and then is incapable of making correct predictions afterwards. An example of what could cause this would be a system being trained to recognize faces being trained with 1000 images of a single person’s face as being faces and a wide array of other objects as not faces. If the testing face is not the person the system was trained with, it would be unlikely to recognize it because it was trained too much with the training face.

Before implementing the pedestrian detection for video, we first tested multiple combinations of feature extraction and classification to see which one would be most suited to our needs based on a combination of prediction accuracy and computation time. The accuracy of each combination is required because without an accurate prediction model we would have the same amount of success if we took chunks of video and randomly assigned a prediction to it. The reason we need to consider computation time when choosing a feature extractor and classifier is that for a video on disk we may have as much time as we need to process it, but for any application which requires speed or detection in real time such as CCTV we require a method that takes at worst 500ms per prediction for a 2 fps video.

**2. Training**

**2.1 Pre-Processing**

While training images are being loaded, they are being pre-processed to improve results and reduce the time taken to run the classifiers. The testing images are also being processed in the same manner before being tested against each model. The images are being pre-processed in 2 main ways:

* Grayscale Conversion
* Normalisation

Grayscale conversion is simply converting each of the images used from RGB to 255 shades of gray. As part of this process, each image is reduced from having 4 colour channels (red, green, blue, and alpha) to only 1 (luminance) which cuts the amount of memory required by approximately 75% and improves the speed of comparing images by approximately the same amount. While some information may be lost in this conversion, the effects go mostly unnoticed as most of the image’s information is stored in the luminance plane which is retained in grayscale while the chrominance planes, which hold far less information, are lost. This can be seen below in figure 1. The top left quadrant shows the full RGB image, the bottom left shows the luminance plane (grayscale) and the two on the right show the dropped chrominance planes.



Figure - Luminance and Chrominance

The second method used is normalization which simply consists of converting the grayscale image into a set of double values. This changes the range of intensity values from the grayscale image from 0..255 to 0..1. To normalise the images the MatLab built in function im2double[2] is used. This function also converts the 0..1 value to fill the entire dynamic range of the image.

Another pre-processing method that was tested was gamma correction which has the goal of balancing brightness across all of the given images to correct any that may be too bright or too dark compared to the rest of the set. Unfortunately this had a noticeable negative impact on accuracy for almost every feature extractor and classifier combination, up to a 7% decrease as seen in table 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Extractor/Classifier | No Gamma | Gamma | % Decrease |
| NN Raw | 0.71278 | 0.67778 | 4.91 |
| NN HOG | 0.71889 | 0.66333 | 7.73 |
| NN PCA | 0.77778 | 0.72056 | 7.36 |
| KNN9 Raw | 0.71778 | 0.72778 | -1.39 |
| KNN9 HOG | 0.65722 | 0.63944 | 2.71 |
| KNN9 PCA | 0.73667 | 0.73278 | 0.53 |
| SVM Raw | 0.80722 | 0.74778 | 7.36 |
| SVM HOG | 0.76778 | 0.76000 | 1.01 |
| SVM PCA | 0.76778 | 0.76000 | 1.01 |
| Adaboost Raw | 0.74278 | 0.74278 | 0.00 |
| Adaboost HOG | 0.81889 | 0.79889 | 2.44 |
| Adaboost PCA | 0.71444 | 0.70778 | 0.93 |

Table 1 - Accuracy with and without gamma correction for sample rate 25

**2.2 Feature Extraction**

The following subsections contain brief summaries of the feature extraction methods that were tested during the development of the pedestrian detector.

**2.2.1 Raw Pixels**

A “feature extractor” with nothing exciting. This is simply using the images as they are loaded in our classifiers. As each image is a 2 dimensional matrix of floats, it can be used as a set of numbers which can be compared to other sets of numbers. This extractor tends to have poor accuracy and long processing times making it a very poor choice.



Figure - Sample of raw images

**2.2.2 Dimensionality Reduction / PCA**

Dimensionality reduction is the process of transforming a data from a high-dimensional space to a lower dimensional space. The benefits of dimensionality reduction include reduced processing time after the overhead of the initial reduction and reduced memory requirement. It also makes data clearer to view as it can reduce huge matrices to tiny ones very quickly. As an example, our raw images our 96\*160, but after applying PCA they are reduced to only 1\*29.

The method of dimensionality reduction that was chosen was PCA (Principal Component Analysis) because it is a well-known general purpose dimensionality reduction technique which doesn’t require much effort to implement.

PCA works by extracting the main components of a dataset and reducing them to a lower dimension. An example is shown below. Using dimensionality reduction, the 2d oval of triangles can be reduced to a 1 dimensional line and still have most of its relevant data (distance between triangles) left intact.

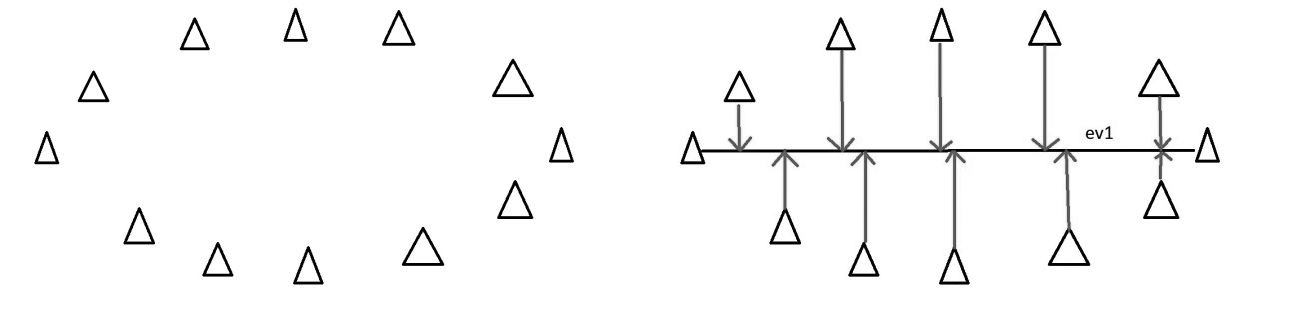


Figure - Simple dimensionality reduction with triangles[3]

One major downside to PCA is the initial overhead of reducing our training images to a lower dimensionality. This is counteracted by scaling each of the training and testing images to half their size before applying PCA. This also means that any further testing images would also need to be scaled down to half size as well.

PCA’s results tend to be found very quickly, but are not quite as accurate as HOG.

**2.2.3 HOG**

The Histogram of Oriented Gradients is a feature extraction method designed for use with pedestrian detection[4]. It works by dividing images into series of cells and then computing the gradients of pixel directions across each cell. After the gradients are found, several stages of normalization are then performed after which the cells are combined back together. An example of what the HOG looks like for Lena is shown below in figure 4.

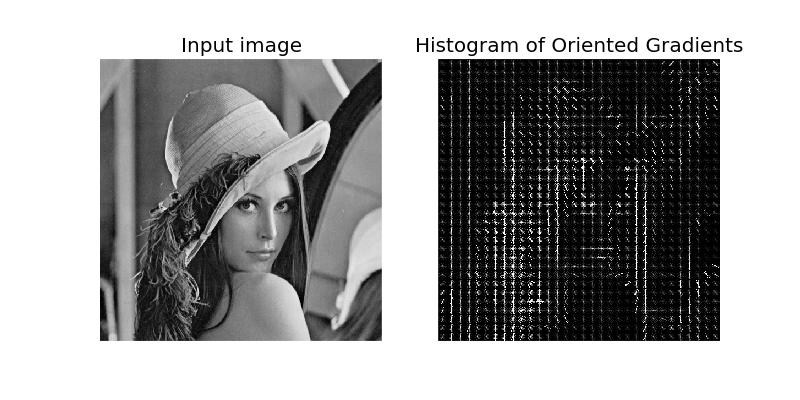


Figure - Lena and her Histogram of Oriented Gradients

HOG has several advantages to it. Since it operates on local cells, it is invariant to geometric transformations such as rotation and photometric transformations such as blur, and because of these properties HOG permits movement of pedestrians to be ignored so long as they remain mostly upright and is therefore particularly suited for human detection in images.

During testing HOG was found to be a more accurate classifier than PCA, but also a slower one.

**2.3 Classification**

The following subsections contain brief summaries of the classification methods that were tested during the development of the pedestrian detector.

**2.3.1 Blind**

This implementation is only technically a classifier. It does not consider the state of the objects it is provided with when developing its model, nor does it when asked to classify an object. Instead, it simply assigns every object it is given to classify a positive value. In practise, it serves more as an indication of how a given split of the dataset looks, as the accuracy value it gives is equivalent to the number of positively labelled objects in the test set. It is largely useless as a classifier in practise, as the values it gives are as useful as a random classifier.

**2.3.2 Nearest Neighbour and K-Nearest Neighbour**

KNN is a classifier which predicts a result by finding the K objects in the model which are closest to the testing data and returning the most common result from the model. An example is shown below in figure 5.

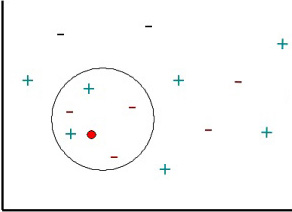


Figure - Nearest Neighbour example

In this example, the red dot is the testing image and the surrounding + and – symbols represent positive and negative labels from the training model. For a single nearest neighbour, this system would return a positive value for this test because the closest single object is positive. If KNN is then used, with a value of K=5, a negative value would be returned instead. This is because the 5 closest objects contain 2 positive and 3 negative values, so the most common value is negative and that will be the prediction given. Using KNN instead of NN has the benefit that outliers tend to be ignored. In the example above, if the further positive value was also a negative it would be clear that the dot belongs to the negative group, however single NN would return positive despite that single positive being an outlier.

NN and KNN are very simple classifiers to implement and can give reasonably accurate results, especially with larger datasets. However, they also come with the serious drawback that as training dataset grows the time taken to retrieve a prediction also grows at the same rate, leading them to be the slowest classifiers tested. This is especially noticeable when combined with the raw image feature extractor where the full set of training and testing images for NN took 84 minutes to complete.

**2.3.3 SVM**

Support Vector Machines is a supervised learning mode where the algorithm is supplied with a set of training samples, each marked as belonging to a category, and will then attempt to assign new samples to one of those categories. The SVM model is a representation of the samples as points in space, mapped in such a way that the examples of the categories are separated by a clear gap. In figure 6 below, the left side shows a set of balls in red and blue. On the right hand side a clear line separating the two categories is visible. New samples would then be tested and placed into one side or the other, and be given that side’s value as its prediction.

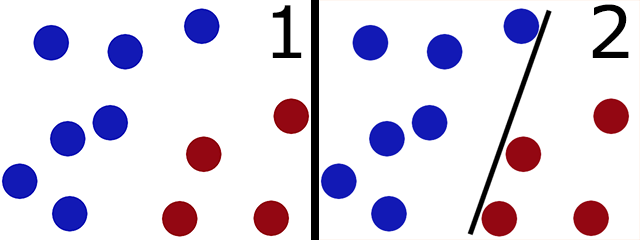
****

Figure - 2d SVM example

In the event that there is a mix between values, or for a non-linear space, SVM can use a kernel function to convert the data to a higher dimensional space before calculating the boundary. An example of this can be seen in figure 7. There is no straight line which can divide red and blue, so the system is converted to a 3 dimensional space where a plane can separate them. For higher dimensions a hyperplane can be constructed to separate them. A further example of this is available at reference 5[5].

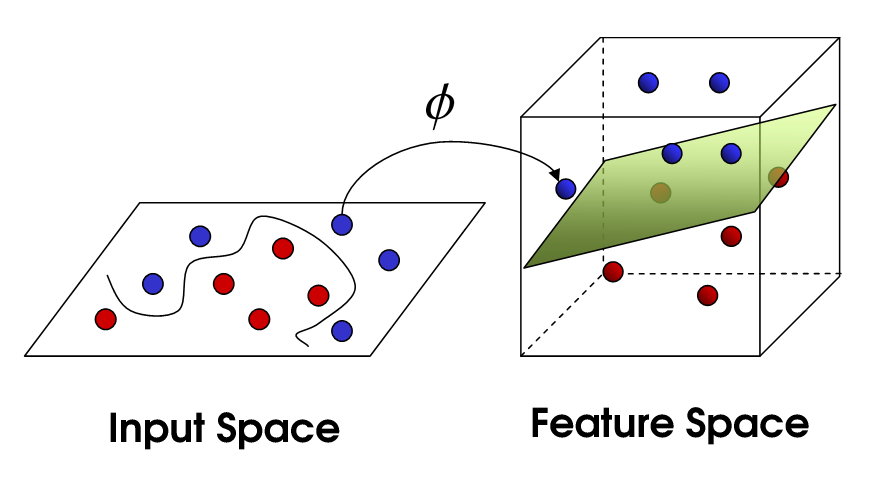


Figure - SVM for non-linear partitions

SVM is a good classifier which works well in practice and works very quickly during the testing stage. It also guarantees an optimal solution for a dataset[6]. However it does take a long time to build the training model. During the testing part of this project, SVM was a consistent good contender for being the chosen model based on both speed and accuracy.

**2.3.4 AdaBoost**

AdaBoosting, or Adaptive Boosting, is a boosting machine learning technique. As a boosting technique, it is a supervised learning mode, and consists of multiple weak learners. This is in contrast to techniques such as SVM, which consist of a single arbitrary strong classifier. A weak learner is any classifier that shows only marginal correlation with the dataset – that is, is typically only slightly better than randomly guessing the label of a sample.

To generate each weak learner of the model, the model evaluates many weak classifiers to separate the samples, which are equally weighted to begin with. From the generated weak samples, it selects those that most accurately reflect the labels of the training set. Each misclassified sample then has their weight increased. The process is then repeated the desired number of times, with all subsequent classifiers being chosen to attempt to classify the newly weighted sample set as accurately as possible and then re-weighting the now misclassified sample. A visualisation of this process across multiple steps is shown in figure 8. By building a composite of these weak learners, the accuracy of the combined weak learners will converge to a high accuracy and thus a strong learner.

For this implementation, a decision tree is used as the weak learner. The resulting performance of the system indicates a long time for generating the model given the number of decision points used to generate it. However, we have chosen this number of points for high accuracy, and classifying a sample with the model is still quite fast. Thus, by pre-generating the model, the performance with this classifier is good enough that it can be used in real-time purposes. The model itself can have issues with outlier values and noise in the training set, but is very unlikely to be prone to overfitting. Additionally, the results generated for large sample datasets show this classification to consistently have one of the highest degrees of accuracy of the tested classification implementations.

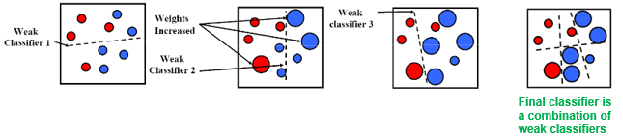


Figure - Visualisation of the AdaBoost training process

**2.3.5 Composite Classifier**

This particular method of classification is not, by itself, a classifier. It instead, as the name implies, is a composition of multiple classifiers with weights, and works by determining both the results from each classifier and the confidence the classifier has in the result it chose. For each classifier it consists of, it generates a model, using the same training data with each of them. It then stores the resulting set of models to use as its own model. Similarly, it iterates through the corresponding testing methods of the classifiers and then adjusts the confidences they produce by the weight the classifier was given. It uses this value to represent this classifier’s vote in what the result is. The result with the highest total confidence value from all classifiers is the result the composite classifier returns, while the final confidence value is calculated based on level of disagreement in the weighted votes.

This classifier has the overhead of every constituent classifier as well as the additional overhead of calculating the final result, and it suffers at least partially from the problems with each of the components. However, the results generated typically, depending on the weighting values used, result in accuracy at worst equivalent to the least accurate of the classifiers, while being capable of outstripping the accuracy of all sampled classifiers. This is, however, unlikely, as one set tends to dominate the others by demonstrating higher confidences.

**2.4 Parameter Tuning**

During the testing phase several sets of parameters were checked in an attempt to find a more accurate set of values. These are briefly explained below.

**2.4.1 PCA Dimensionality**

As PCA changes the way data is viewed by giving the eigenvectors that best represent the data, it is possible to choose a set of eigenvectors to work on which provide the greatest variance. In the code for this project, that is found as the second parameter of the applyPCA method. As changing the number of dimensions to retain changes the results, there was some evaluation of accuracies before settling on a final value for the number of dimensions. The results of this experimentation are found below in table 2.

|  |  |  |
| --- | --- | --- |
| Classifier | Dimensions | Accuracy |
| SVM | 236 | 0.8321 |
| SVM | 101 | 0.8521 |
| SVM | 48 | 0.8494 |
| SVM | 25 | 0.8381 |
| AdaBoost | 236 | 0.855 |
| AdaBoost | 101 | 0.8654 |
| AdaBoost | 48 | 0.8608 |
| AdaBoost | 25 | 0.8534 |

Table 2 - Values for PCA dimension testing

The numbers of dimensions chosen were the numbers which represent keeping 95% of data, 90%, 85%, and 80% respectively. These values were tested with both the SVM and AdaBoost classifiers and the final value found to be used is 101, or 90% of data. While this isn’t a huge change, it is still a relevant value.

**2.4.2 Adaboost Learning Cycles**

Owing to the nature of Adaboost, that of generating a number of weak classifiers and compositing them, both the accuracy and performance of the function depends on the number of classifiers created. Through experimentation, the effects of varying this parameter were examined, and this was used to choose the number of classifiers generated. This is shown below. Note that these results were cross-validated with ten iterations using a 1:1 split.

|  |  |  |  |
| --- | --- | --- | --- |
| Pre-processing | Learning Cycles | Accuracy (%) | Time taken (s) |
| Raw | 1 | 74.5333 | 0.80835 |
| Raw | 10 | 76.1333 | 9.4399 |
| Raw | 20 | 81.1333 | 17.8625 |
| Raw | 50 | 82.4667 | 48.1654 |
| HOG | 1 | 71.4 | 0.42253 |
| HOG | 10 | 82.1333 | 5.0243 |
| HOG | 20 | 85.1333 | 9.2188 |
| HOG | 50 | 86.4 | 25.5235 |
| PCA | 1 | 69.6667 | 0.038643 |
| PCA | 10 | 74.3333 | 0.19138 |
| PCA | 20 | 77.3333 | 0.37197 |
| PCA | 50 | 80.2 | 0.89701 |

Table 3 - Values for Adaboost Learning Cycles

As can be seen, each increase in the number of learning cycles increases the time by a similar factor – jumping from 1 cycle with raw to 10 cycles with raw results in an 1167% increase in time taken, with a 2% increase in time. The bulk of this time taken, as explained in the description of AdaBoost, is owing to the cost of generating models. From this, it is clear that, given sufficient time to pre-generate the model, the classifier would approach a perfect detector. Owing to time constraints, 50 has been selected as the number of learning cycles, as it results in a sufficiently high accuracy rate while also not being exceptionally prohibitive in terms of time taken to generate.

**3. Testing**

**3.1 Data Split**

**3.1.1 Half Split**

For initial testing and checking that the system was working, the data was split evenly into training and testing sets. This is useful for simple tests and validation, and it runs quickly, but it does not have the accuracy or reliability of the cross validated splits that were used when deciding upon a feature extractor/classifier combination for detection.

**3.1.2 Cross Validation**

In addition to using a simple 50:50 split for the data to analyse the performance of the classifiers, a number of cross validated runs were performed. This means that a number of permutations of the dataset were generated, and each was used across multiple different ratios of training data to testing data. This allows for comparison of the relative performances of more and lesser trained classifiers, as well as giving an idea of how the general case of a classifier performs. Note that one of the ratios used in Cross Validation was that of a 1:1 ratio – that is, a half split. This was to consolidate the analysis of the performance of both metrics. For choosing the final model to use, many cross validated runs were performed and the results analysed. The model that consistently produced the most accurate results from these tests was the one ultimately used.

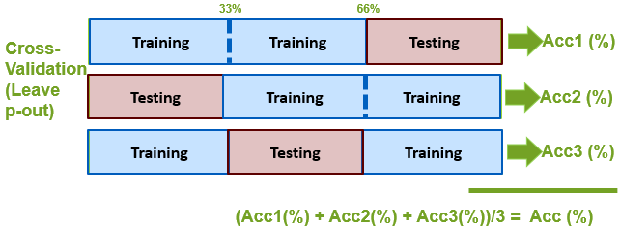


Figure - Visualisation of a simple Cross Validation example

**3.2 Evaluation**

**3.2.1 Recognition Rate**

Table 2 below shows the accuracies retrieved by testing each of the developed feature extractors and classifiers using a sampling rate of 1. These are the cross-validated results using 3 ratios and 8 iterations. The results obtained here and the results obtained by re-running the code may be different due to the cross-validation method randomising the training and testing sets.

|  |  |  |
| --- | --- | --- |
| Extractor/Classifier | Base Accuracy | Time Taken |
| NN Raw | 0.64944 | 5050.1588 |
| NN HOG | 0.90478 | 2517.1057 |
| NN PCA | 0.84889 | 112.1943 |
| KNN9 Raw | 0.67438 | 5231.5712 |
| KNN9 HOG | 0.91829 | 2586.7554 |
| KNN9 PCA | 0.84396 | 129.5293 |
| SVM Raw | 0.80942 | 364.1680 |
| SVM HOG | 0.97031 | 112.6426 |
| SVM PCA | 0.97031 | 118.3255 |
| Adaboost Raw | 0.86127 | 686.1594 |
| Adaboost HOG | 0.94418 | 627.9432 |
| Adaboost PCA | 0.86478 | 9.4895 |
| Blind Raw | 0.66632 | 1.9536 |
| Blind HOG | 0.66632 | 1.6908 |
| Blind PCA | 0.66632 | 1.5281 |
| Composite (Adaboost + SVM) Raw | 0.85968 | 1185.1275 |
| Composite (Adaboost + SVM) HOG | 0.94783 | 842.8450 |
| Composite (Adaboost + SVM) PCA | 0.86646 | 139.1543 |

Table 4 - Accuracy for all extractor/classifier combinations with sample rate 1

From the table, the most accurate classifier is SVM with a 97% detection rate for both HOG and PCA, making it the best choice of classifier for accurate pedestrian detection.

Also to note is that the raw images were consistently the worst performers in terms of accuracy, and took the longest amount of time to complete by a large amount. This is especially noticeable when combined with the Nearest Neighbour classifier where it took approximately 84 minutes to complete.

The PCA results were fast across the board with a maximum running time of 129 seconds for KNN-PCA. Given their accuracy is higher than that of raw images this makes them reasonable contenders as the choice for the video detection feature extractor.

As seen above, SVM-HOG and SVM-PCA are the most accurate classifiers with a 97% accuracy rate each. This is mostly expected as SVM is a strong classifier which provides optimal partitioning for datasets and the HOG feature extractor was designed for pedestrian detection while the PCA feature extractor reduces the testing and training data to a small set of prominent components.

**3.2.2 Type 1 and 2 Errors**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Classifier |  | FP | FN | Precision | Sensitivity | Specificity | FPR | F-Measure |
| Blind | Raw | 50.000 | 0.000 | 0.667 | 1.000 | 0.000 | 1.000 | 0.800 |
| NN | Raw | 9.000 | 34.000 | 0.880 | 0.660 | 0.820 | 0.180 | 0.754 |
| KNN3 | Raw | 15.000 | 26.000 | 0.831 | 0.740 | 0.700 | 0.300 | 0.783 |
| KNN9 | Raw | 22.000 | 19.000 | 0.786 | 0.810 | 0.560 | 0.440 | 0.798 |
| Fuzzy KNN9 | Raw | 22.000 | 19.000 | 0.786 | 0.810 | 0.560 | 0.440 | 0.798 |
| Fuzzy KNN9 HW | Raw | 13.000 | 26.000 | 0.851 | 0.740 | 0.740 | 0.260 | 0.791 |
| Fuzzy KNN9 LW | Raw | 22.000 | 19.000 | 0.786 | 0.810 | 0.560 | 0.440 | 0.798 |
| SVM | Raw | 4.000 | 33.000 | 0.944 | 0.670 | 0.920 | 0.080 | 0.784 |
| AdaBoost | Raw | 17.000 | 12.000 | 0.838 | 0.880 | 0.660 | 0.340 | 0.859 |
| Composite (SVM/AdaBoost) | Raw | 17.000 | 12.000 | 0.838 | 0.880 | 0.660 | 0.340 | 0.859 |
| Blind | HOG | 50.000 | 0.000 | 0.667 | 1.000 | 0.000 | 1.000 | 0.800 |
| NN | HOG | 10.000 | 9.000 | 0.901 | 0.910 | 0.800 | 0.200 | 0.905 |
| KNN3 | HOG | 16.000 | 6.000 | 0.855 | 0.940 | 0.680 | 0.320 | 0.895 |
| KNN9 | HOG | 21.000 | 0.000 | 0.826 | 1.000 | 0.580 | 0.420 | 0.905 |
| Fuzzy KNN9 | HOG | 21.000 | 0.000 | 0.826 | 1.000 | 0.580 | 0.420 | 0.905 |
| Fuzzy KNN9 HW | HOG | 17.000 | 2.000 | 0.852 | 0.980 | 0.660 | 0.340 | 0.912 |
| Fuzzy KNN9 LW | HOG | 21.000 | 0.000 | 0.826 | 1.000 | 0.580 | 0.420 | 0.905 |
| SVM | HOG | 19.000 | 0.000 | 0.840 | 1.000 | 0.620 | 0.380 | 0.913 |
| AdaBoost | HOG | 16.000 | 4.000 | 0.857 | 0.960 | 0.680 | 0.320 | 0.906 |
| Composite (SVM/AdaBoost) | HOG | 16.000 | 4.000 | 0.857 | 0.960 | 0.680 | 0.320 | 0.906 |
| Blind | PCA | 50.000 | 0.000 | 0.667 | 1.000 | 0.000 | 1.000 | 0.800 |
| NN | PCA | 16.000 | 27.000 | 0.820 | 0.730 | 0.680 | 0.320 | 0.772 |
| KNN3 | PCA | 18.000 | 15.000 | 0.825 | 0.850 | 0.640 | 0.360 | 0.837 |
| KNN9 | PCA | 24.000 | 11.000 | 0.788 | 0.890 | 0.520 | 0.480 | 0.836 |
| Fuzzy KNN9 | PCA | 23.000 | 13.000 | 0.791 | 0.870 | 0.540 | 0.460 | 0.829 |
| Fuzzy KNN9 HW | PCA | 17.000 | 24.000 | 0.817 | 0.760 | 0.660 | 0.340 | 0.788 |
| Fuzzy KNN9 LW | PCA | 23.000 | 11.000 | 0.795 | 0.890 | 0.540 | 0.460 | 0.840 |
| SVM | PCA | 15.000 | 7.000 | 0.861 | 0.930 | 0.700 | 0.300 | 0.894 |
| AdaBoost | PCA | 17.000 | 11.000 | 0.840 | 0.890 | 0.660 | 0.340 | 0.864 |
| Composite (SVM/AdaBoost) | PCA | 17.000 | 11.000 | 0.840 | 0.890 | 0.660 | 0.340 | 0.864 |
| Blind | AVG | 50.000 | 0.000 | 0.667 | 1.000 | 0.000 | 1.000 | 0.800 |
| NN | AVG | 11.667 | 23.333 | 0.867 | 0.767 | 0.767 | 0.233 | 0.811 |
| KNN3 | AVG | 16.333 | 15.667 | 0.837 | 0.843 | 0.673 | 0.327 | 0.839 |
| KNN9 | AVG | 22.333 | 10.000 | 0.800 | 0.900 | 0.553 | 0.447 | 0.846 |
| Fuzzy KNN9 | AVG | 22.000 | 10.667 | 0.801 | 0.893 | 0.560 | 0.440 | 0.844 |
| Fuzzy KNN9 HW | AVG | 15.667 | 17.333 | 0.840 | 0.827 | 0.687 | 0.313 | 0.830 |
| Fuzzy KNN9 LW | AVG | 22.000 | 10.000 | 0.803 | 0.900 | 0.560 | 0.440 | 0.848 |
| SVM | AVG | 12.667 | 13.333 | 0.882 | 0.867 | 0.747 | 0.253 | 0.864 |
| AdaBoost | AVG | 16.667 | 9.000 | 0.845 | 0.910 | 0.667 | 0.333 | 0.876 |
| Composite (SVM/AdaBoost) | AVG | 16.667 | 9.000 | 0.845 | 0.910 | 0.667 | 0.333 | 0.876 |

Table 5 - Error Results

Note that the results of Table 5 were obtained with a sample rate of 10 using a 50:50 split and a single iteration of training and testing for each classifier. As can be seen from analysis of the average FN and FP variables, or from the Sensitivity and Specificity, SVM and AdaBoost have noticeably less errors on average than all other classifiers analysed, with some minor exceptions that occur due to the specific permutation of the data.

Analysis of the average F-Measure of the different pre-processing techniques shows that HOG has, on average, the superior performance. Using the observation obtained by analysis of the averages of each set, that of SVM and Adaboost having the best detection rate, only those two are worth consideration for this technique. Analysis of all the error metrics leads to the conclusion that SVM with HOG has the lowest error rate as a whole. Even without reducing the choices to two variables gives this result, as SVM with HOG has the highest F-Measure value of all tested classifiers, and has exactly zero false negatives.

**3.2.3 ROC Curves**

ROC (Receiver Operating Characteristic) Curves are charts that illustrate the performance of binary classifiers as their discrimination thresholds vary. They are created by plotting the true positive rate against the false positive rate at varying thresholds of confidence. The area under the ROC curve represents the accuracy of the testing method. This means that the closer the ROC curve is to the top and left of the chart, the more accurate the testing method is and the closer the curve falls to the 45 degree angle the less accurate the testing method is.

The ROC curves for the SVM-HOG, SVM-PCA, AdaBoost-HOG, and AdaBoost-PCA combinations are shown below in figure 10.

As seen in the examples, both AdaBoost and SVM have lower accuracies, bordering on a 45 degree angle. However the curve for SVM-HOG is significantly higher, as we expect to see given our cross-validated SVM-HOG method being the most reliable testing method.



Figure - ROC Curves samples.

**3.3 Reflection**

**3.3.1 Method Comparison**

As stated in section 3.2.1 the most accurate combinations of feature extractor and classifier are joint between SVM-HOG and SVM-PCA at 97% accuracy. The only difference between the two combinations is a 6 second timing difference.

Table 3 below shows the feature extractors and classifiers ranked in order of both speed and accuracy.

PCA is reliably the fastest feature extractor while HOG is the most accurate. Using the raw image is both the slowest and least accurate, ruling it out as a choice for the detection algorithm entirely. For the classifiers, the blind classifier is by far the fastest however it is approximately equal in performance to a random classifier and mainly gives insight to the split of positive and negative images. The second fastest classifier and the most accurate classifier is SVM.

|  |  |  |  |
| --- | --- | --- | --- |
| Feature Extractor Speed | Feature Extractor Accuracy | Classifier Speed | Classifier Accuracy |
| PCA | HOG | Blind | SVM |
| HOG | PCA | SVM | Composite |
| RAW | RAW | AdaBoost | AdaBoost |
|  | | Composite | KNN |
| NN | NN |
| KNN | Blind |

Table 6 - Speed and Accuracy tables

Using the information found from these comparisons, the chosen feature extractor and classifier combination used for detection is SVM-HOG. SVM-HOG was chosen because it and SVM-PCA are the most accurate classifiers by 3% over AdaBoost HOG while running in approximately one fifth of the time, and SVM-HOG runs approximately 5% faster than SVM-PCA.

**3.3.2 Known Failures**

Running the cross-validation for SVM-HOG returns the following 63 image indexes as being consistently mislabelled: 89, 137, 147, 223, 226, 242, 254, 265, 270, 278, 294, 295, 321, 347, 473, 486, 670, 683, 706, 718, 760, 780, 839, 918, 1146, 1174, 1246, 1278, 1382, 1468, 1479, 1522, 1534, 1569, 1580, 1582, 1634, 1678, 1702, 1794, 1825, 1837, 1869, 1881, 1905, 1916, 1942, 1956, 2035, 2099, 2148, 2190, 2323, 2353, 2398, 2505, 2528, 2598, 2676, 2740, 2808, 2896, and 2960.

50 of these images are shown below in figure 11. Approximately 21 of these images are mostly one uniform colour. This leads to the conclusion that the reason for the SVM-HOG classifier to fail is that it has trouble predicting that blank backgrounds are not people in some cases. One potential reason for this is that a lot of the positive images contain people against featureless backgrounds such as beaches, snow, grass, and pavement. This reason that this could be effecting the predictions is that there may not be enough positive images containing people against busier backgrounds for the system to learn from.

It is also interesting to note that none of the consistently mislabelled images are false negatives, they are all false positives. This means that the SVM-HOG system is slightly overzealous when it comes to predicting positives and is consistently better at not accidentally stating that people are not people.

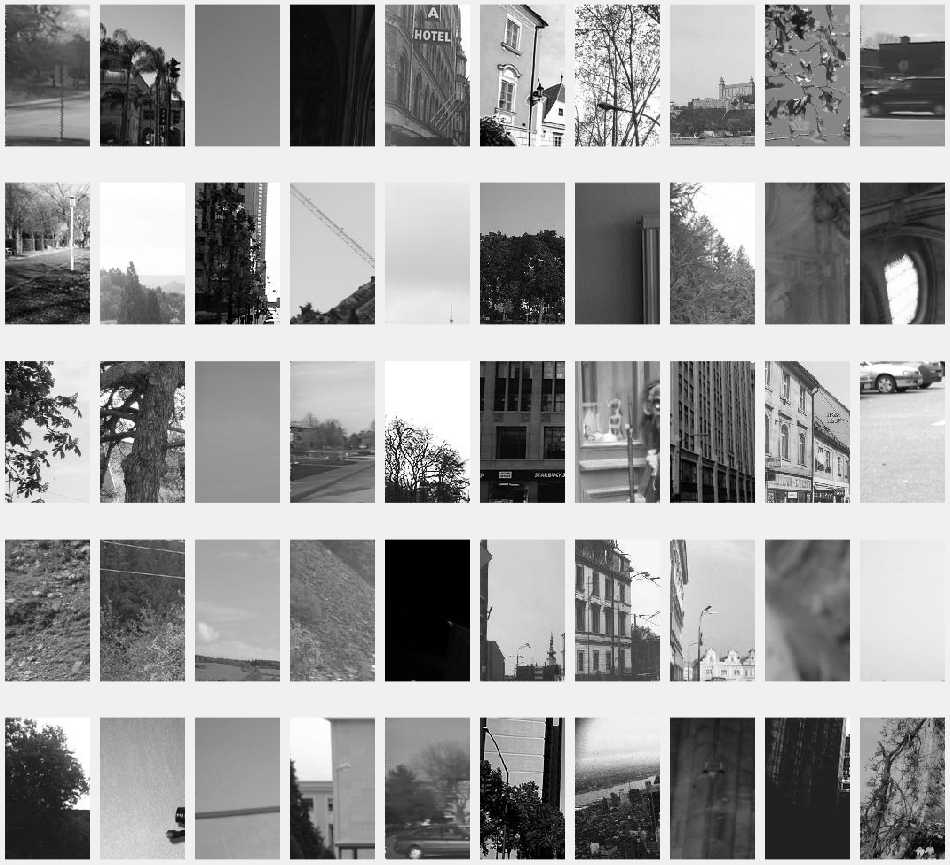


Figure 11 - Consistently mislabelled images for SVM-HOG

**4. Detection**

**4.1 Chosen Model**

Based off section 3.3.1 and the above results, SVM with HOG has been shown to be the most accurate classifier with a slightly faster execution speed than SVM and PCA, leaving it as the best classifier for the purposes of this project.

**4.2 Sliding Window**

Due to the difference in size between the training images and the video, using an entire video frame as a testing image is not a valid way to check for the presence of a person. Instead, each frame of the video is cut down into chunks and those chunks are used as the testing images. This is done using a sliding window which moves across the frame like a typewriter and cuts out pieces of the frame. The result of this process can be seen in figure 11.



Figure - Sliding Window example

This still leaves the problem of having people in the distance appear smaller in the video, and people nearby appear larger. To combat this, a multi-scale sliding window is used. Instead of making a single pass over the video, multiple passes are used with different window sizes. For this system the sizes used are the training image size, 1.5 times the training image size, and 0.5 times the training image size. The general cutout of the video for all three frame sizes can be seen in figure 12.

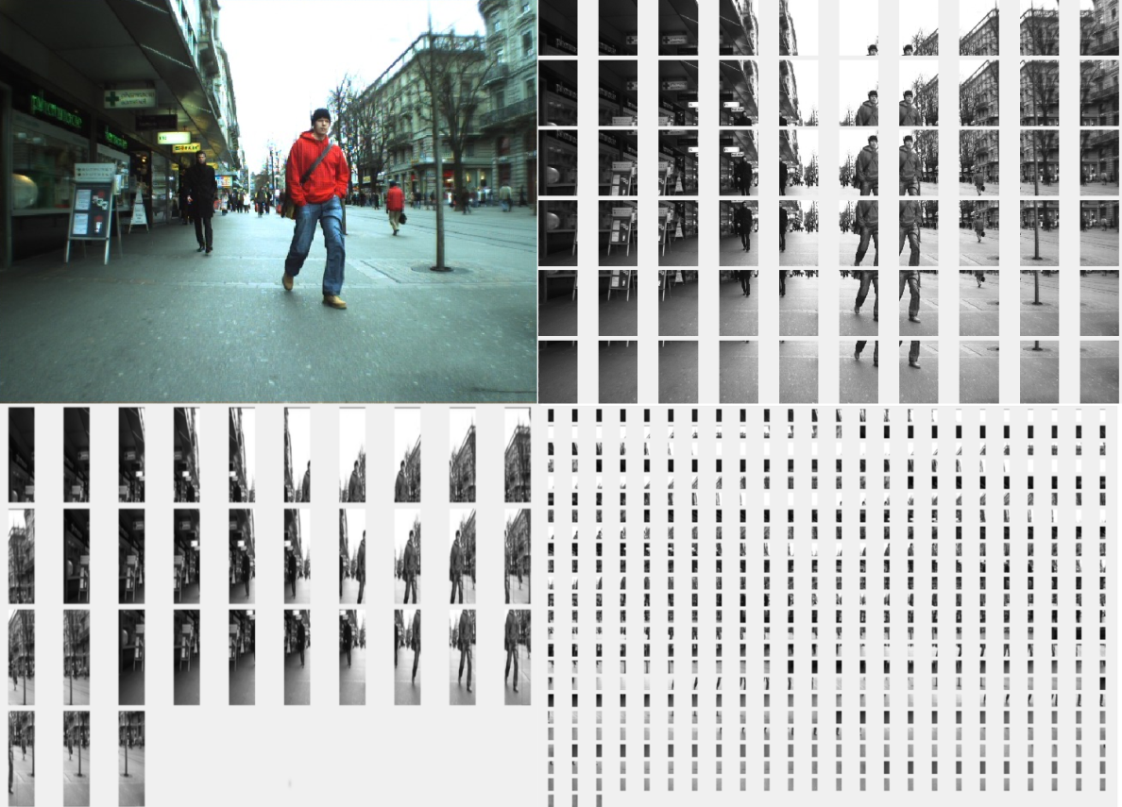


Figure - Multi-Scale Sliding Window Example

**4.3 Non-Maxima Suppression [4]**

As shown in figure 12, due to the segmenting of the image we will get multiple positive results produced for the same image area. The algorithm places bounding boxes around the areas where positive results were predicted and based off confidence values some of those are culled, but there will still be some overlap. Using NMS removes most of this overlap by checking intersecting boxes against each other, and when a significant overlap is found, discarding boxes with lower confidence values. This leaves the most confident guesses for each area. The results of this can be seen in figure 13. The left side of figure 13 shows the video after NMS has been applied while the right side of the image shows all of the areas where a person was predicted to be.

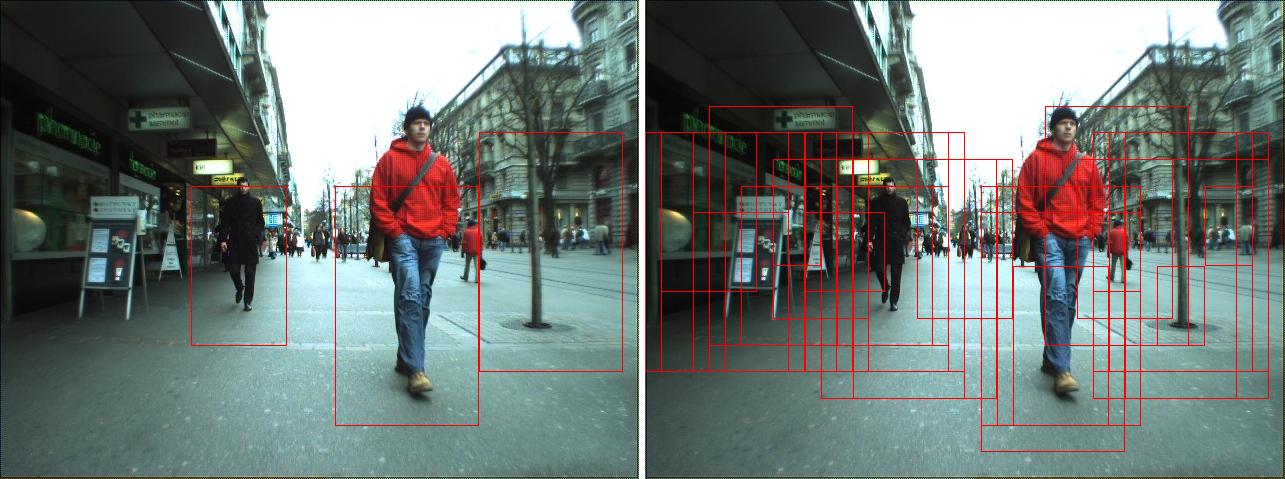


Figure - Non-Maxima Suppression Example

**4.4 Results**

**4.4.1 Comparison to test.dataset**

Show a few screencaps of our detection vs the testing one. We need to get test.dataset drawn on top of the video for this. I can do that ezpz.

**4.4.2 Known Failures**

Which frames or people are we missing? Do visual inspection vs test.dataset to see if we can spot anyone who stands out. All those tiny people in the background probably don't count.

**5. Testing Notes**

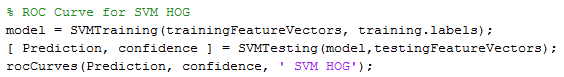
**5.1 Sampling Rate**

The sampling rate for execution is modifiable at the top of the main script. Using a sampling rate of 1 should only be done with time to spare. When choosing the most accurate combination a sampling rate of 10 was used however for the output videos provided a sampling rate of 10 was used as the time taken for sampling rate 1 was too high for repeated processing.



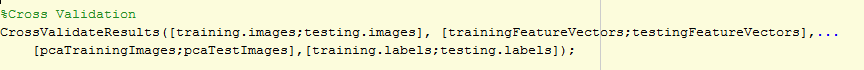
**5.2 ROC Curves**

ROC Curves are generated by creating a model, testing the model, and calling the rocCurves method.



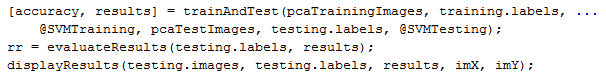
**5.3 Cross Validation**

The cross validated results can be viewed by uncommenting these lines in the main script.



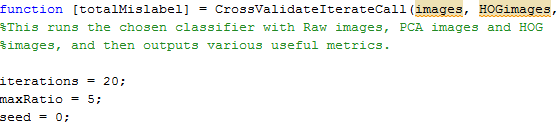
**5.4 Type 1 and 2 errors, sensitivity, precision, etc, and viewing of correct and incorrect images**

These are viewable with the following lines of code (example for SVM-PCA)



**5.5 Changing Cross-Validation Iterations**

The number of iterations and ratios can be changed in CrossValidateResults->CrossValidateIterateCall.



**6. References**

[1]"xkcd: Electoral Precedent", *Xkcd.com*, 2016. [Online]. Available: <https://xkcd.com/1122/>

[2]"Convert image to double precision - MATLAB im2double", *Uk.mathworks.com*, 2016. [Online]. Available: <http://uk.mathworks.com/help/matlab/ref/im2double.html>

[3]"Principal Component Analysis 4 Dummies: Eigenvectors, Eigenvalues and Dimension Reduction", *George Dallas*, 2013. [Online]. Available: <https://georgemdallas.wordpress.com/2013/10/30/principal-component-analysis-4-dummies-eigenvectors-eigenvalues-and-dimension-reduction/>

[4]N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection", *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*.

[5]"SVM with polynomial kernel visualization", *YouTube*, 2016. [Online]. Available: <https://www.youtube.com/watch?v=3liCbRZPrZA>

[6]”Analysis of SVMs” [Online]. Available: <https://learning.qol.qub.ac.uk/2151/CSC/4067-SPR-QUB/_layouts/15/WopiFrame.aspx?sourcedoc=/2151/CSC/4067-SPR-QUB/Resources/Computer%20Vision/4-Machine%20Learning_alumni.pptx&action=default>