CSC3061 Group Assignment

REPORT

Contents

Jonathan Kernaghan (40175824)
jkernaghan03@qub.ac.uk

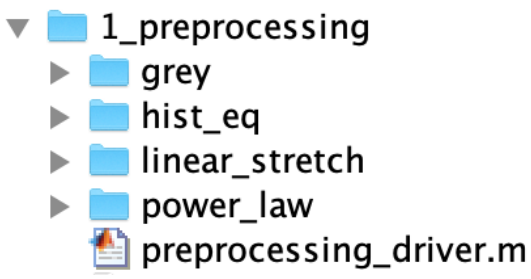
Matthew Elliott (40153557)
melliott21@qub.ac.uk

David Bradley (40029212)
dbradley19@qub.ac.uk

https://gitlab2.eeecs.qub.ac.uk/CSC3061-1920/csc3061-1920-g2

A user guide is in the Git repository in order to assist in understanding how to operate our solution. If any issues arise, please get in contact with one of us. Our final video is also in the repositories: models/optimised_baseline folder

Preprocessing Overview

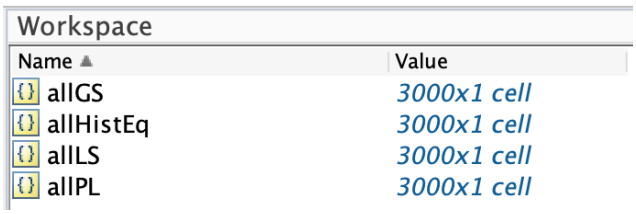



```
% Set up environment variables for preprocessed images here
clc;
path = 'images/**/*.jpg';
%-----------------------------------GREY_SCALE-----------------------------------%

[allGS] = grey_scale(path);

%-----------------------------------HIST_EQ-----------------------------------%

[allHistEq] = hist_eq(path);

%-----------------------------------POWER_LAW-----------------------------------%

[allPL] = power_law(path);

%-----------------------------------LINEAR_STRETCHING-----------------------------------%

[allLS] = linear_stretch(path);
```

We have chosen to implement *all three major preprocessing methods* highlighted throughout the course. Additionally, we include greyscale as a baseline comparison in order to exhibit the improvements offered visually and quantitively by each contrast enhancement.

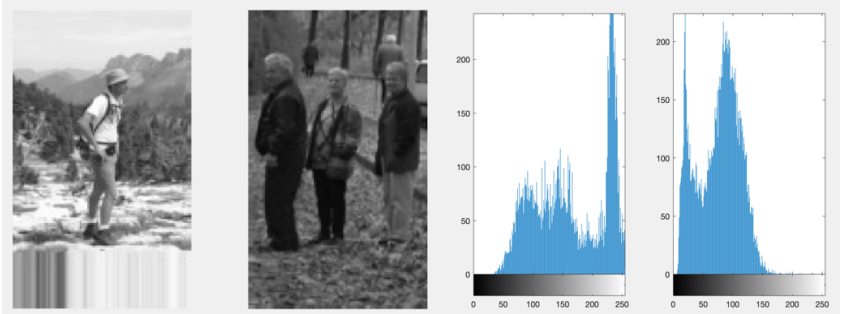
Setting up the environment variables i.e. the arrays containing each pre-processed image is simple – *Open 'preprocessing_driver.m' and click run.*

This has been designed to limit excess variables in the environment by encapsulating them within functions as such:



Why do we think it is worthwhile considering all three contrast enhancement techniques?

Since we are working with a dataset of 3000 images, it is important to note that there is an inherently large variance of dynamic range. This is due to differences in the image quality, lighting, environmental differences etcetera. For example, after converting to grayscale –



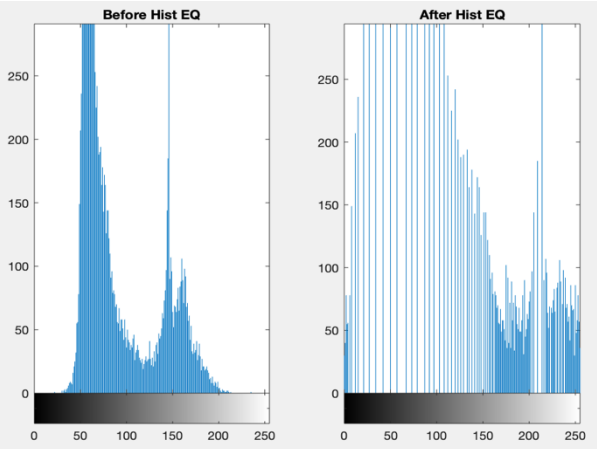
Therefore, when selecting a technique for contrast enhancement we must choose one which is generalised, and offers the best quantitative results when feature extracted with HOG and trained/tested.

It is not a good solution to apply multiple techniques to different groups of the dataset for which we deem them appropriate. This would lead to us creating a solution which is too specialised to the conditions of the training data – a form of overfitting in terms of preprocessing at least.

Histogram Equalisation with Visual Analysis

Histogram Equalisation is a contrast enhancement technique with no input parameters. The intensities, i.e., the high-density peaks are distributed more evenly across the histogram, leading to better contrast in low contrast areas.

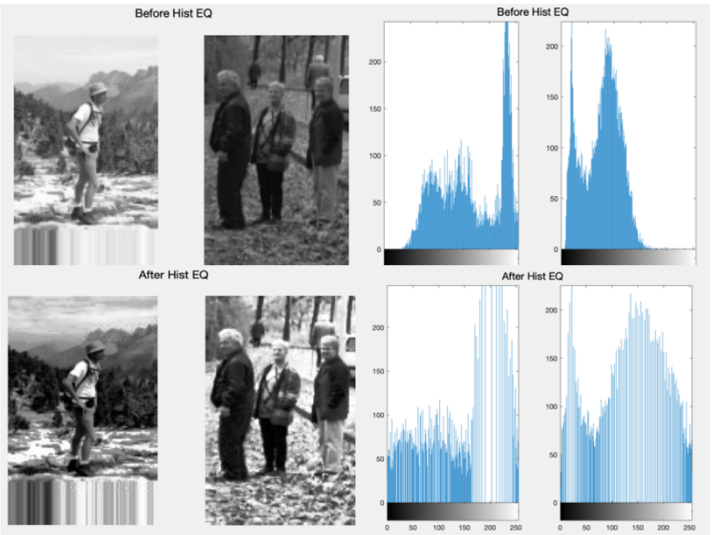
Let us take the following example of a relatively low contrast image –



We can see that our implementation of histogram equalisation has done a good job of increasing the contrast visually and this is reflected in the histogram. It has also increased the overall brightness of the image by making use of the full dynamic range.

Histogram Equalisation *might* be a prime candidate for our solution based on the fact that it requires no parameters and is not specialised for particular contrast enhancement problems. However, it seems to produce harsh, oversaturated and therefore unnatural looking images. Though perhaps this is a problem only to the human eye and might be beneficial when it comes to feature extraction.

Taking the earlier conflicting examples which exhibit variance in terms of brightness/contrast –



We can see that visually; *the left-hand image has probably reduced in quality* to the human eye. It is harder to see the individual details of the man's face, arms & lower torso.

The *right-hand image has increased in quality*, we can see a greater distinction between foreground (people) and background (foliage). However, it has dealt with these two extremes fairly well, despite the left-hand loss.

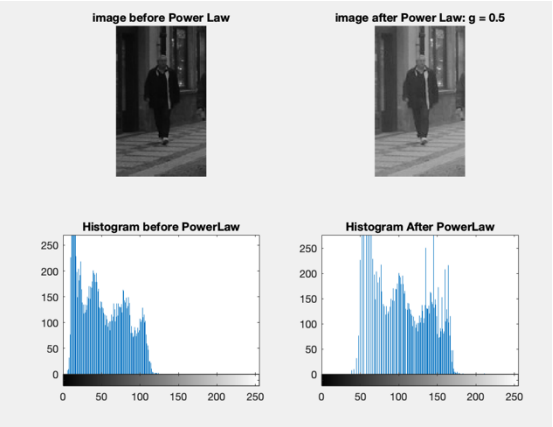
Power Law with Visual Analysis

Power law is used to enhance the contrast in an image. it uses the “gamma” input in order to determine whether the image has to increase in brightness or decrease in brightness. Image pixel is run against the following Power Law formula:

$$O = \text{round}\left(\frac{I^{\gamma}}{255^{\gamma-1}}\right)$$

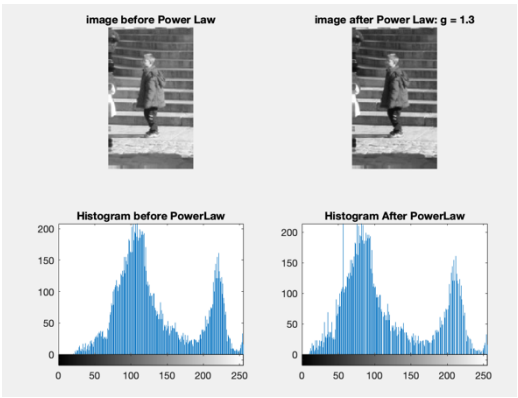
This formula takes the original value of the pixel, applies the formula and the predetermined gamma value and gives out the new updated pixel value.

To get the gamma value, the image was first converted to greyscale, then the average value of the pixels within the whole image was found. If the average value was below 125, then the gamma was set to 0.5. this number was chosen as it is below 1. And if the average was above 125, then the gamma was set to 1.3.



As the image shows, with the gamma set to 0.5, the contract and brightness of the image has increased. The background is now more visible and the contract between the person in the middle of the image and their surroundings is more evident. The Histograms also show that the shift towards the higher greyscale, showing the increase in brightness, with more spread towards the bottom values on the scale.

The image to the right shows one of the training images that had its contrast and brightness reduced. The histograms also show the shift in scale. The image itself has sharper edges around the person in the middle. With his jacket being more obviously different than the stairs behind him.



With Power Law being able to dynamically determine the contrast change for each individual image. It is probable that this would be the best suit for pre-processing for both the training images, and the final video images.

Linear Stretching with Visual Analysis

Linear stretching (California Institute of Technology, n.d.) was performed on all 3000 training images. To combat random noise in the image on the highest and lowest pixel values not allowing for a full stretch, the ten pixels with the highest and ten pixels with the lowest pixel values were ignored.

Results

Visually to the human eye, linear stretching had mixed results. Generally, the level of enhancement of a linearly transformed image is positively correlated with the gradient of the look up table that did the transformation

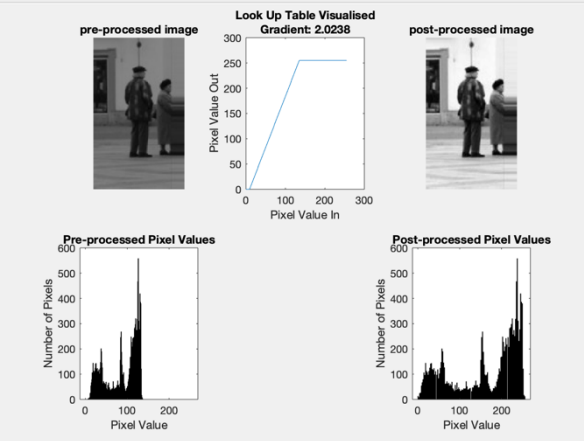


Figure 1: A linear stretch transformation with a look up table gradient of 2.02.

The steep gradient of 2.02 enhances the detail, particularly the 2 people in the image.

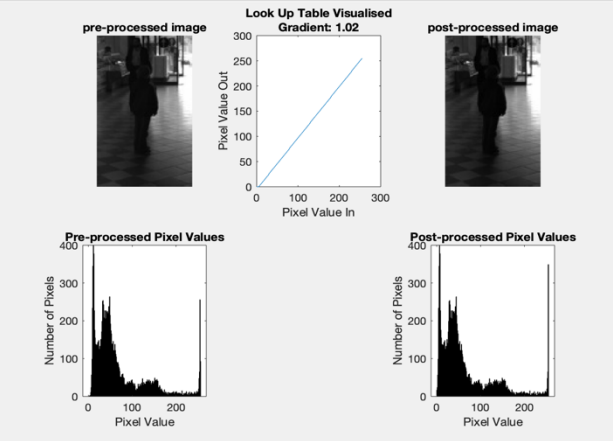


Figure 2: A linear stretch transformation with a look up table gradient of 1.02

The look up table gradient is almost at its minimum, 1, resulting on almost no transformation of the image.

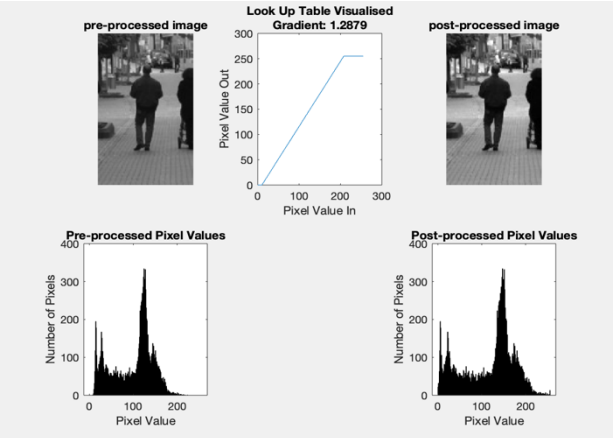


Figure 3: A linear stretch transformation with a look up table gradient of 1.29.

The transformation is just about visible to the human eye. It was theorized that a look up table transformation gradient below 1.3 would not yield any noticeable effects on the image.

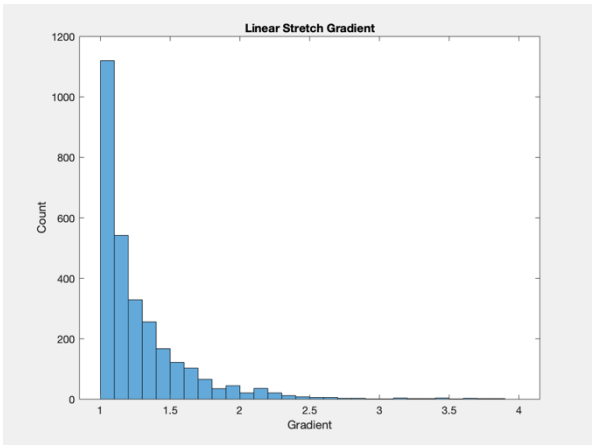


Figure 4: Graph of the count of look up table gradients over the 3000 supplied train images.

It can be derived that that 68.1% of images have a gradient under 1.3. This would suggest that linear stretch will only be effective on 31.9% of images.

Without testing, it is impossible to know if this will negatively affect the results later on. Perhaps most of the images that have a linear stretch gradient below 1.3 are already sufficient for feature extraction.

Greyscale

In order to have a baseline in which the three preprocessing techniques described above could be compared to, the greyscale function was created. It simply takes in all of the images and uses the rgb2gray function (MathWorks, n.d.).

Using this function, it can be verified whether the preprocessing has a positive or negative effect and to what extent alters the accuracy of the learning methods being used.

Feature Extraction using Histogram of Oriented Gradients (HOG)

Histogram of Oriented Gradients is a feature descriptor whereby the orientation of gradients within localised regions of an image can be measured in order to define the presence of an object.

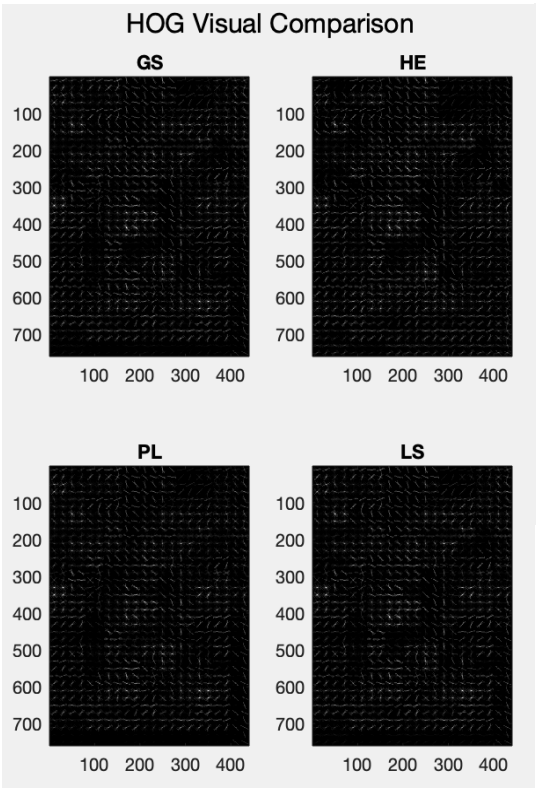
Why is HOG suitable for pedestrian detection?

Physiologically, a pedestrian has several identifiable features/components that can be utilised for detection. For example – the head, arms, and legs etcetera. Instead of elaborately extracting these features from pixel neighbourhoods representing edges, HOG encapsulates the shape of the person as a whole.

Haar-like features are an example of edge extraction.

Example of Histogram of Oriented gradients as above shows the scope to include other features - like the shoulder, or items of clothing

In our specific case, we have selected HOG as the feature descriptor because it successfully encompasses all possible features of the pedestrian in a generalised way that can be easily deciphered by the model. However, it is not always aesthetically pleasing to the human eye – especially in our context where there is a lot of background noise and other objects in the images. For example, examine the following –



We cannot see the outline/shape of this lady in the HOG visualisations presented on the left. There is no discernible difference between the baseline grayscale HOG and the contrast enhanced HOG's. Certainly, *we would be unable to conclude quantitively which enhancement is superior right now.*

Potential Learning Model Candidates - SVM, RF & NN/KNN

Here we outline the three proposed machine learning model candidates for the problem of pedestrian detection –

Support Vector Machine (SVM)

We have chosen to implement SVM as it is a well-documented classifier with several kernels for an optimal hyperplane dependent upon dataset making it a flexible candidate. It is suitable for small to medium size datasets which makes it particularly viable for us. However, the three kernels and subsequent parameters lead to a relatively time-consuming tuning.

Random Forest

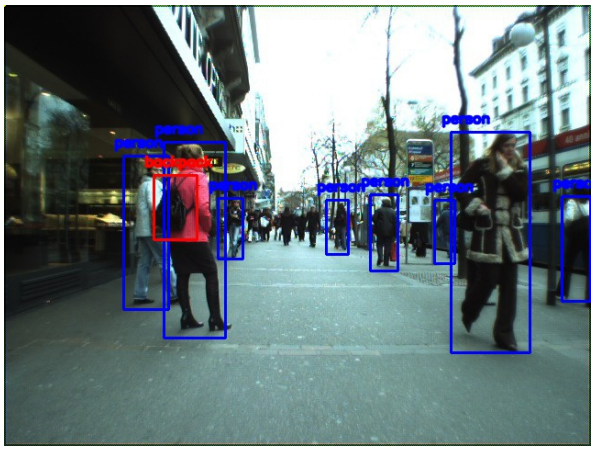
One of the key benefits of random forest is how easy it is to use - due to its small number of parameters. It can handle a range of features naively of preprocessing. It is inherently binary so may be well suited to our problem domain.

Nearest Neighbour/K-Nearest Neighbour (KNN)

We have chosen to implement KNN as it quite easy to code and flexible enough to produce decent results with a medium size dataset. Finding the optimal K value shouldn't be too laborious compared to the hyperparameter tuning of SVM. However, it is a computationally intensive operation to search for nearest neighbours and consumes a lot of memory.

Cognitive Neural Network (CNN)

Based on previous experiences with machine learning, a pre-existing library for python known as "Darkflow"(1) was used as a test bed, to see how an established detector would fair with the provided frames. A frame from this is shown below.



While this detection was very accurate, it unfortunately could not be used as the library was written in python and due to time constraints, was not possible to investigate whether or not this could be implemented in Matlab. It did serve as a guide throughout the project as to how other established detectors would handle the frames.

Ref: (1) Darkflow, https://github.com/thtrieu/darkflow, 2016.

Hyperparameter Optimisations

SVM

Bayesian Optimisation vs Grid Search

An exhaustive grid search is computationally intensive as it requires a high number of parameter combinations. Despite this, it may yield a marginally superior parameter combination than the less intensive Bayesian optimisation. However, the method of Bayesian optimisation is chosen in our case as it makes smart decisions based on experimented combinations leading to less attempts. It was not feasible for us to perform a grid search since our team is operating on older laptops.

Gaussian/RBF vs Linear vs Polynomial

MATLAB's 'fitcsvm' provides the functionality of carrying out Bayesian optimisation for all parameters of each kernel. Therefore, we will go with the kernel and other parameter values it chooses. We applied a logit score transform to ensure all scores that come back are between 0 and 1. We also set it to use parallel on 4 cores – to speed up the search -

```
modelFITSVM = fitcsvm(images, labels, 'ScoreTransform', 'logit',
'OptimizeHyperparameters', 'all',
'HyperparameterOptimizationOptions',
struct('MaxObjectiveEvaluations', 180, 'UseParallel', true))
```

After 180 evaluations, Bayesian optimisation found the following to be best -

```
C = 37.894;
sigmakernel=3.465;
kernel='rbf';
```

KNN

The K-nearest-neighbors (KNN) algorithm (Wikipedia, n.d.) was considered. Multiple different models were trained and tested using 5-fold cross validation. All odd neighbor values from 1 to 801 were tested.

The pre-processed images were passed into the KNN algorithm:

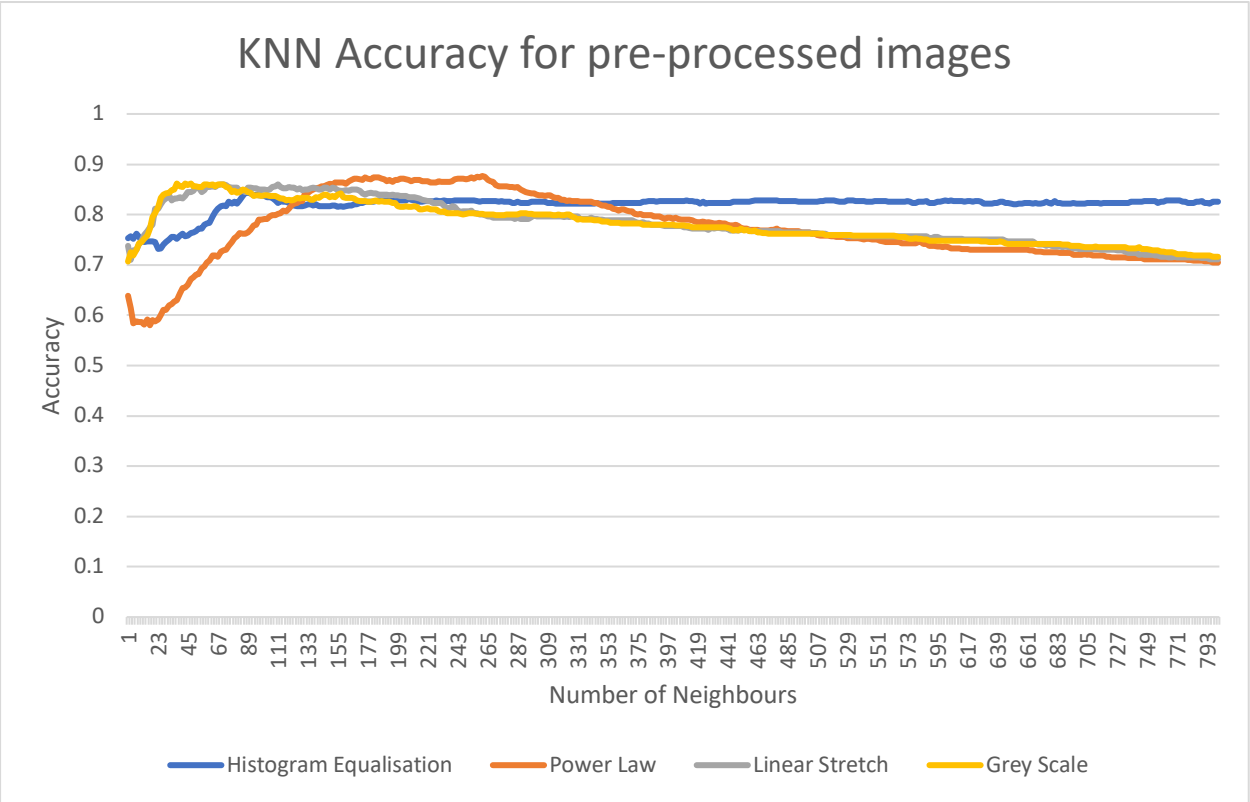


Figure 1.1: 5-fold cross validation accuracy of KNN model under different pre-processing techniques

The pre-processed images were then passed into the HOG feature extractor. The highest accuracy of **0.867** achieved was the image set preprocessed by power law at **265 neighbors**.

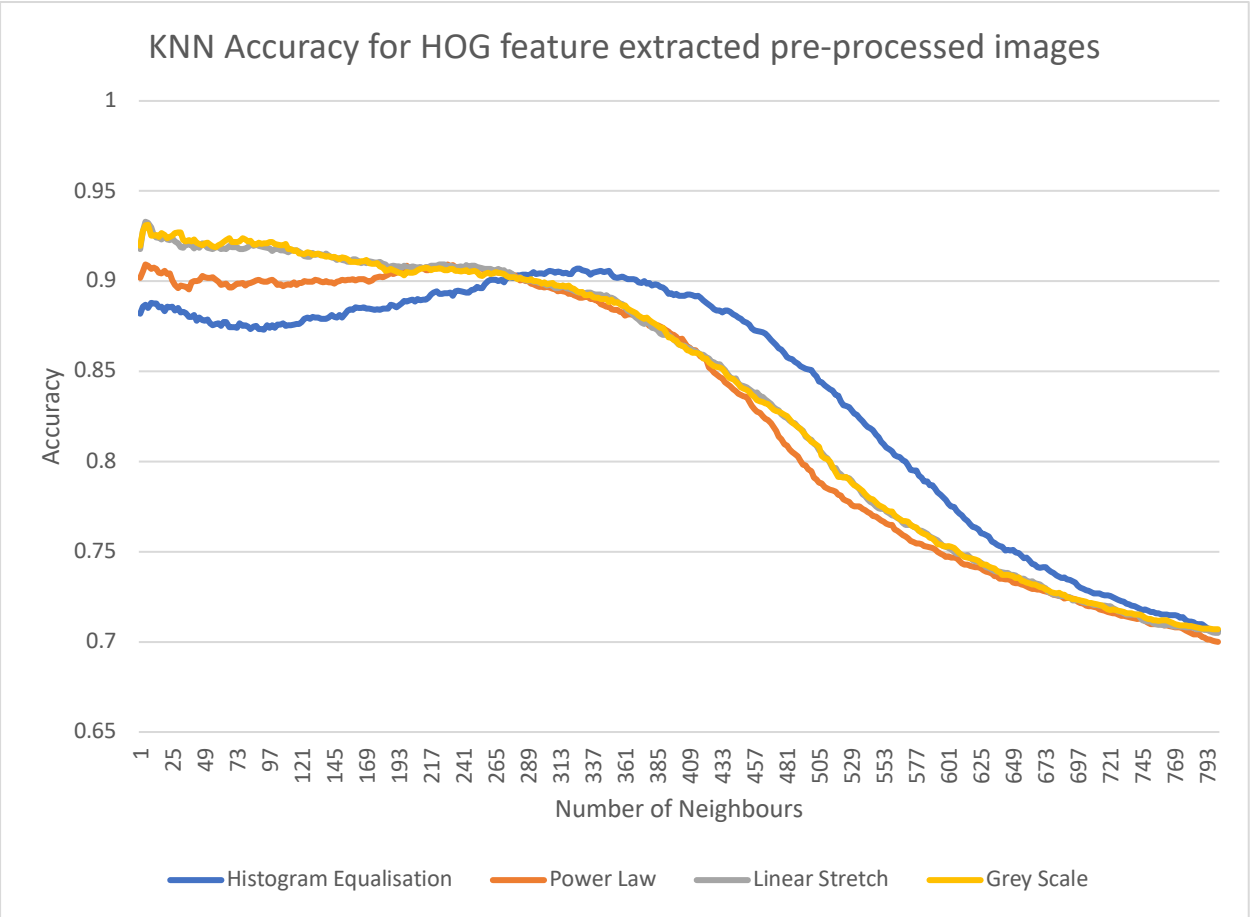


Figure 1.2: 5-fold cross validation accuracy of KNN model under different HOG feature extracted pre-processed image sets

The Highest accuracy of **0.931** was obtained under linear stretch using **11 neighbors** form linearly stretched images that were passed through the HOG feature extractor. It is clear comparing Fig 1.1 to 2 that the HOG feature extractor

Summary: The highest accuracy achieved by KNN was 0.931 using 11 neighbors.

Random forest

Multiple Random Forest models (Wikipedia, n.d.) were trained using the supplied 3000 images to learn to identify a person was in an image. The TreeBagger library (MathWorks, n.d.) was used to construct the model.

To identify the number of trees that would yield the best combination of maximum accuracy without unnecessary numbers of trees being used, the model has the '*OOBPrediction*' flag activated. To measure the accuracy of the model, oobPredict method (MathWorks, n.d.) was applied to the model after it was constructed. This measures the out-of-bag error (Bhatia, 2019). The error values were converted to accuracy values by preforming one minus the error value.

The combination of the '*OOBPrediction*' flag and then using the oobPredict method is used to see how many trees produce what level of accuracy. To get a fair comparison of the accuracy relative the accuracy results of KNN and SVM, the accuracy will have to be measured in the same way: 5-fold cross validation. Therefore, the parameters and data that form the most accurate model will be used in a 5-fold cross validation test.

The out-of-bag accuracy of the random forest model using the raw images was calculated. The model created was specified to have 300 trees and out of bag sampling. All 3000 preprocessed images of each type were passed into the model. See the results in the graph below:

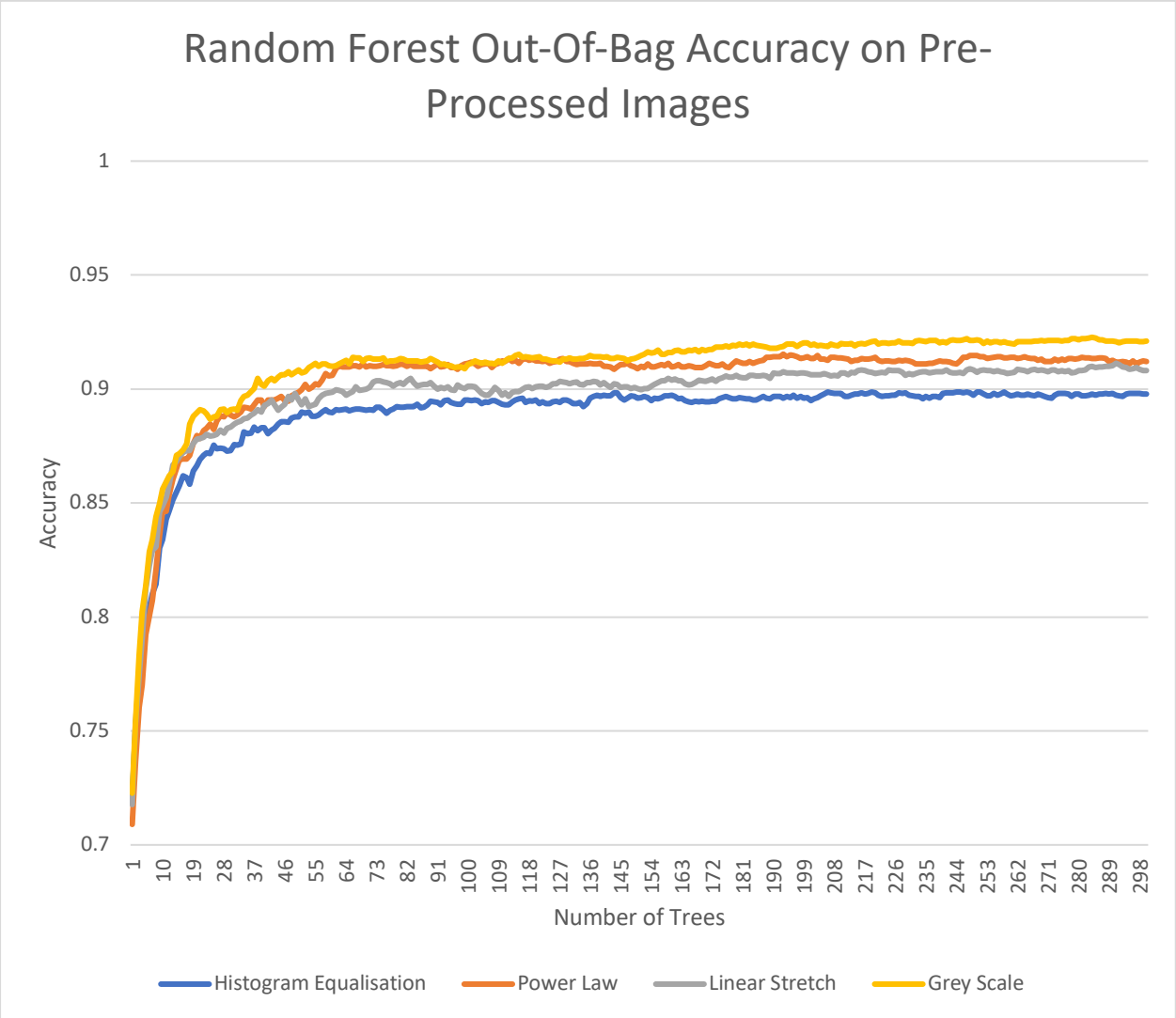


Figure 2.1: Random Forest Out-Of-Bag Accuracy on Pre-Processed Images

From Fig2.1, the highest accuracy of 0.922 was obtained by using the grey scale preprocessing at 287 trees.

As a general trend most of the accuracy of the random forest is gained at 100 trees. Increasing the number of trees beyond 200 has a negligible effect on the accuracy. All pre-processing techniques had a negative effect on the accuracy of the model.

The out-of-bag accuracy of the random forest model using the output of the hog feature extractor on all types four types of pre-processed image was calculated. The model created was specified to have 300 trees and out of bag sampling. All 3000 HOG images of each type were passed into the model.

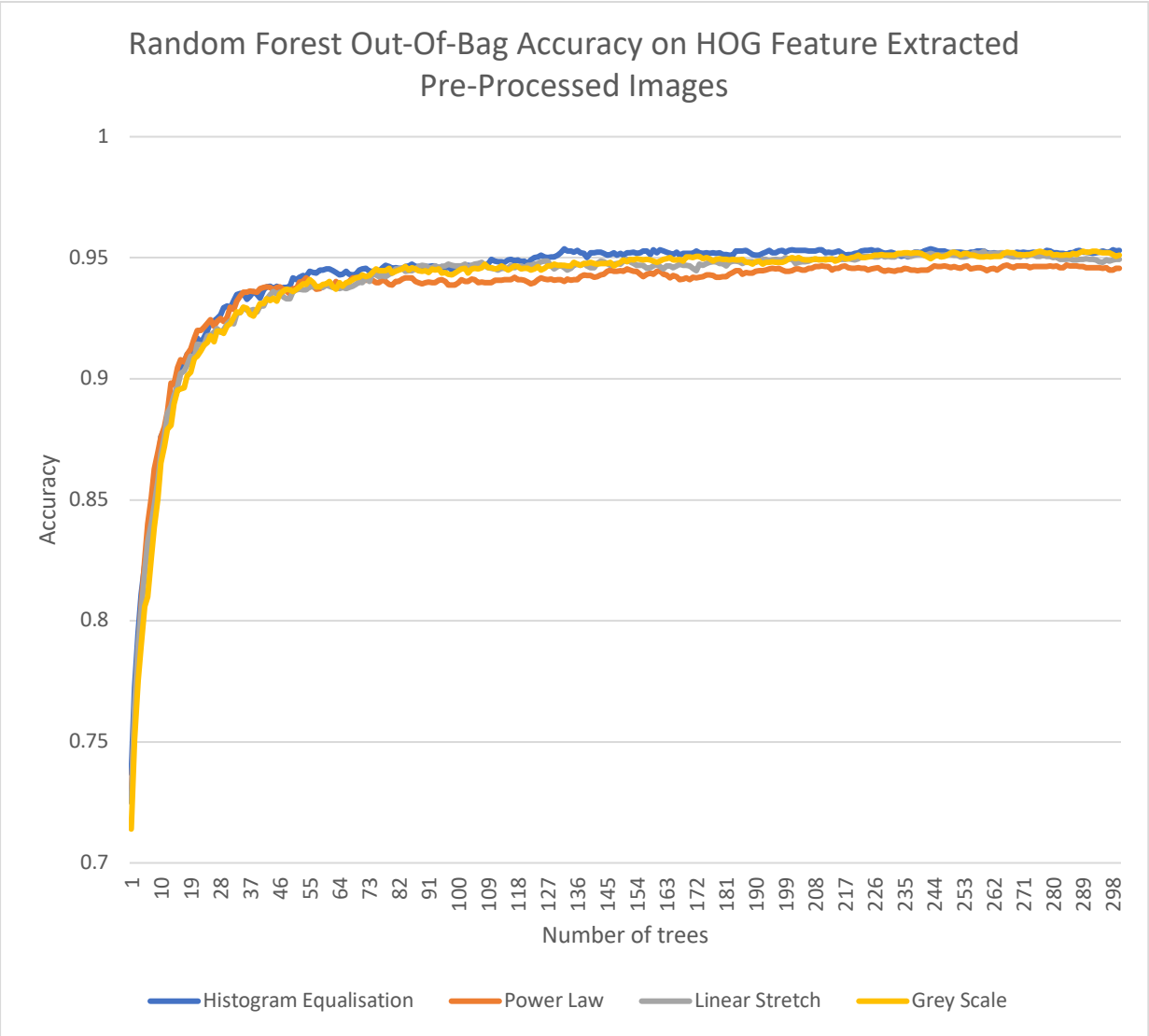


Figure 2.2: Random Forest Out-Of-Bag Accuracy on HOG Feature Extracted Pre-Processed Images

From Fig 2.2, it is seen that the optimal parameters were histogram equalized images passed through a HOG feature extractor. These parameters produced an accuracy of 0.945 at 132 trees. Increasing the number of trees beyond this point has a negligible effect of accuracy.

As seen from Fig 2.1 and 2.2 above, the HOG feature extractor has a positive effect on both the accuracy increase by 2.3% (improved from 0.922 to 0.945) as well as an efficacy increase, reducing the number of trees required by 54% (287 trees to 132 trees).

This most accurate model was then passed through a 5-fold cross validation test to assess the accuracy in a consistent way to how it was being measured with KNN and SVM. The 5-fold accuracy came out to be 0.942, almost identical to the out-of-bag accuracy calculated before.

Pretesting Formulation of Hypotheses

Here we present two hypotheses –

Based on visual inspection of the contrast enhancements in terms of output image and its subsequent histograms. We have developed the following hypothesis:

Preprocessing Hypothesis

H_0 *Histogram equalisation yields optimal accuracy*. We default this as null hypothesis due to the fact it is the most widely used contrast enhancement technique with regards to pedestrian detection. It is general purpose, with no parameters and produces somewhat unnatural looking images in its attempt to make full use of the dynamic range – we hypothesise that this will be computationally useful.
H_1 *Power Law yields optimal accuracy*. This is the first alternative hypothesis. We believe that based on the fact that power law requires a dynamic gamma parameter, it will be capable of making an intelligent decision based on bright and dark regions in images.
H_2 *Linear Stretching yields optimal accuracy*. This is the second alternative hypothesis. We do not think this will yield the best results due to the fact it simple evenly spreads/stretches the histogram out naively of whether or not it is a bright or dark region to make use of the dynamic range.

Machine Learning Model Hypothesis

H_0 *SVM yields optimal accuracy*. We place SVM as the optimal classifier in our null hypothesis since it is the most flexible, with a lot of scope for tuning due to extensive kernels and other parameters. It is also commonly used for pedestrian detection and suitable for our size of dataset.
H_1 *Random Forest yields optimal accuracy.* This is our first alternative hypothesis since random forest is easy to use and work with. It is also well suited to our binary task of detecting a pedestrian or not because of its decision tree architecture.
H_2 *K-Nearest Neighbour yields optimal accuracy.* This is our second alternative hypothesis. We are unsure if our training set is large enough to produce high accuracies with this classifier.

Later we will confirm whether or not our hypotheses held up after quantitively testing each model generated from the combinations of contrast enhancement and classifier. To do this we will test using k fold cross validation and interpreting the evaluation metrics.

Core Pathways to Test

Histogram EQ – HOG – SVM	Histogram EQ – HOG – RF	Histogram EQ – HOG – KNN
Power Law – HOG – SVM	Power Law – HOG – RF	Power Law – HOG – KNN
Linear Stretching – HOG – SVM	Linear Stretching – HOG – RF	Linear Stretching – HOG – KNN

Testing the above using k fold cross validation will ascertain with a great deal of confidence which combination provides optimal accuracy in terms of the evaluation metrics. We will then perform dimensionality reduction on the best and retest to ensure the results are still consistent with this change. Dimensionality reduction is beneficial since each hog array is 3000 by 7524.

Testing Overview – Hold-Out and K-Fold

Partitioning
In line with industry standards, to test our models we used an 80/20 training/testing partition. With our overall dataset of 3000 images, this gives 2400 for training and 600 for testing.

Hold-Out
We wrote a function to do one-off tests with to get a general idea of our experiments accuracy without committing to the more time-consuming K-fold cross validation. This simply splits our dataset using the partition and tests it thusly. It generates a confusion matrix and the evaluation metrics (recall, precision, specificity, f measure and false alarm rate).

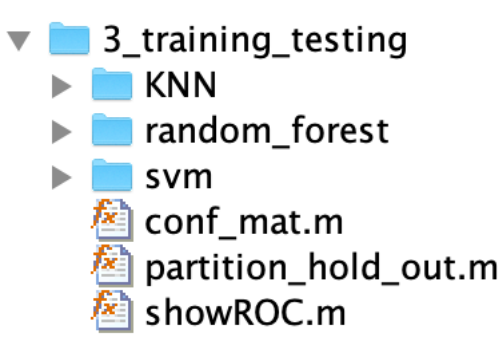
K-Fold Cross Validation
We set our K value to 5. We saw very little intra-fold variance and therefore there was minimal gain to be had by attempting 10 or 20-fold. This was also not technically feasible for us, as would be too computationally intensive. The purpose of cross validation is to minimise the effect of the random element of partitioning, by repartitioning multiple times. To achieve this, we ensured that partitioning was stratified by label – meaning each partition should have similar quantities of each label. Our K-Fold functions also return a confusion matrix and the evaluation metrics (recall, precision, specificity, f measure and false alarm rate).

Confusion Matrices
For each test, either standalone hold-out or k-fold, a confusion matrix will be generated displaying the true positives, false positives, true negatives and false negatives. This provides insight into exactly what our models are doing well and what they are doing badly. However, without some further processing it is hard to properly evaluate the efficiency of the model.

Evaluation Metrics
Using the true positives, false positives, true negatives and false negatives generated by the confusion matrix we will calculate:

Recall – Ratio of correctly labelled pedestrians by the model to all who are actually pedestrians.
Precision – Ratio of correctly labelled pedestrians by the model to all pedestrians labelled.
Specificity – Ratio of correctly labelled negatives by the model to all actually negative
False *Alarm* Rate – Ratio of negatives incorrectly labelled as positive and all actual negatives
F Measure – The harmonic mean of recall and precision

Computation Time
Utilising MATLAB's built-in timer, tic and toc, we will measure the time taken to test our models. This is an important additional metric to consider when evaluating the efficiency of these algorithms.

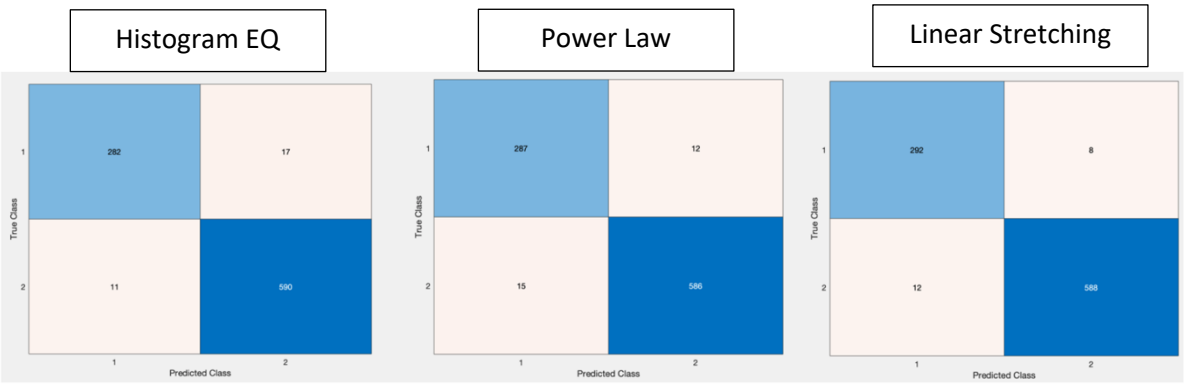


The code for our testing set-ups for each classifier is in the "3_training_testing" folder, alongside some useful utility functions for generating partitions, confusion matrices and the ROC.

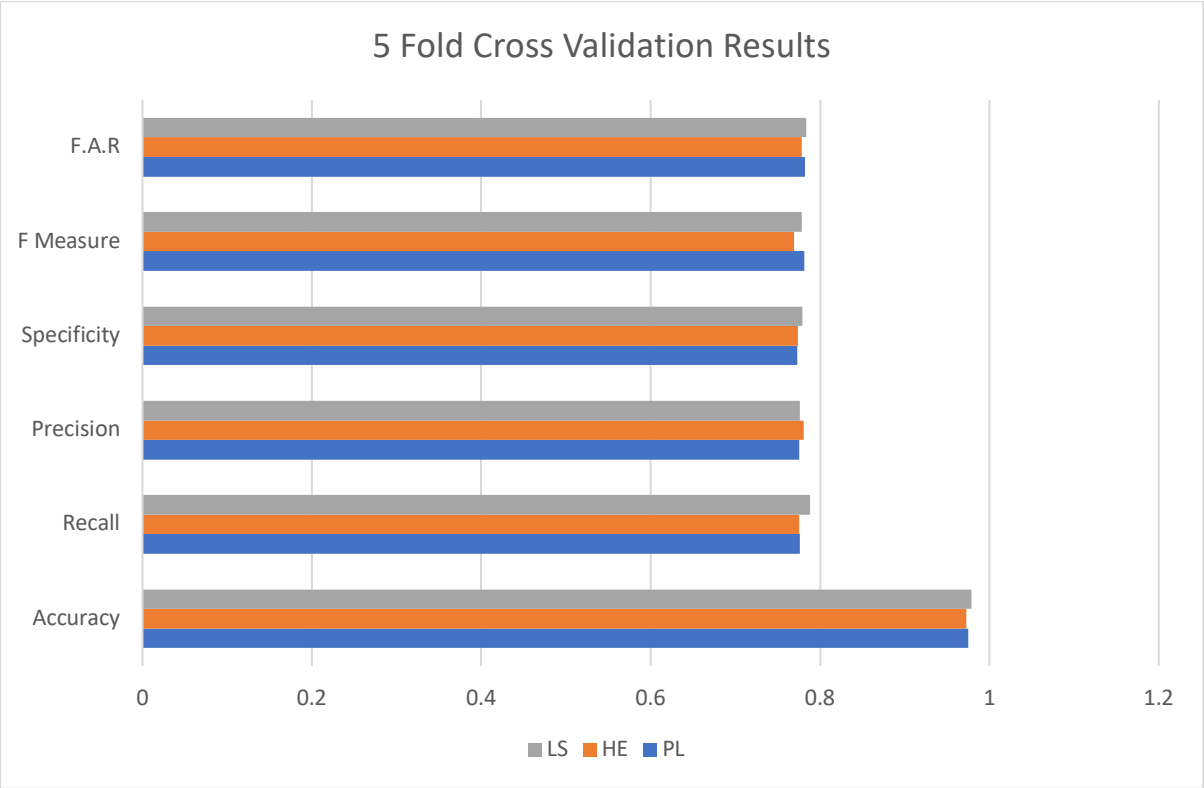
Quantitative Analysis of Cross-Validation Results – Confusion Matrices/Evaluation Metrics

SVM

Below are the three confusion matrices generated by the 5-fold cross validations results for the three contrast enhancement techniques using the HOG-SVM pathway -



Below is the evaluation metrics formed on the basis of the true positives, false positives, true negatives and false negatives from the confusion matrices -



Interpreting the results –

Based on overall accuracy, Linear Stretching is the best. However, this does not take into account the impact of false positives which effectively negate a proportion of those correctly classified. Therefore - we must consider F Measure the more viable indicator of model performance. Based on this *Power Law – HOG – SVM is the optimal model in this comparison with an F measure of 0.78.* This beats marginally beats Linear Stretching which has an F Measure of 0.77.

Random Forest

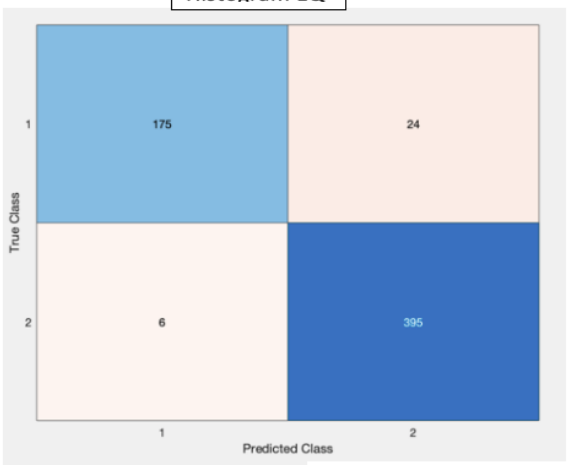
Below are the three confusion matrices generated by the 5-fold cross validations results for the three contrast enhancement techniques using the HOG-RF pathway -

On the top left, and bottom right there lies true negatives and true positives respectively.

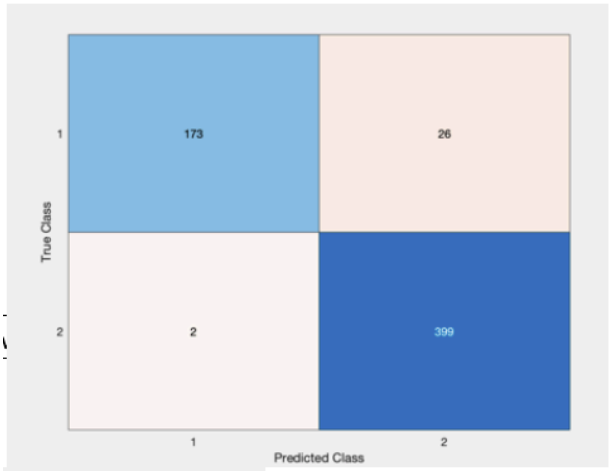
On the Bottom left there lies false negatives and false positivises.

It is clear from evaluating the metrics below that Histogram equalisation is clearly the best pre-processing technique.

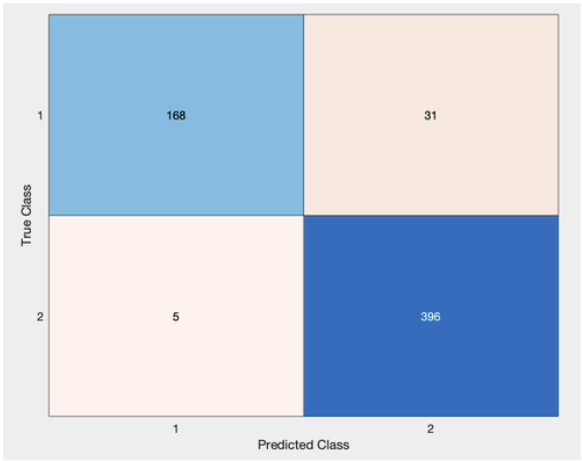
Histogram EQ



Linear Stretching



Power Law



Dimensionality Reduction

The main goal for dimensionality reduction is to improve the accuracy of the models. The time reduction was not going to be a deciding factor whether a form of dimensionality reduction was used, rather just a positive side effect.

The first Dimensionality reduction technique considered was Principal Component Analysis (PCA). The aim is to extract the most dominant features and filter out the noise, both theoretically improving accuracy while reducing computation time. On testing both the random forest and SVM, the number of dimensions were altered to find what level of reduction yielded the most accurate results.

The second Dimensionality reduction technique considered was Linear Component Analysis (LDA). Since there were two classifications (has a person or has not got a person), the hog dimensions passed into the LDA function would be reduced to one feature per image. The fraction of PCA dimensions considered by the LDA was tested between 1 and 0.8. Again, the goal was to see what combinations of parameters yielded the best accuracy.

Only two models were determined to be accurate enough to be used in the video:

1) Random forest using the HOG feature extracted histogram equalized images.
2) SVM using the HOG feature extracted histogram on power law images.

From previous analysis, these two models provide very distinct ways of identifying a person in an image while maintaining an accuracy of above 0.95. This was not achievable by KNN.

<u>Random forest – PCA</u>

As before, the same method of using the random forest with the 'OOBPrediction' flag with the oobPredict method (MathWorks, n.d.).

The forest was trained with 300 trees was tested on every 5th dimension from 1 up to 496. See the results below:

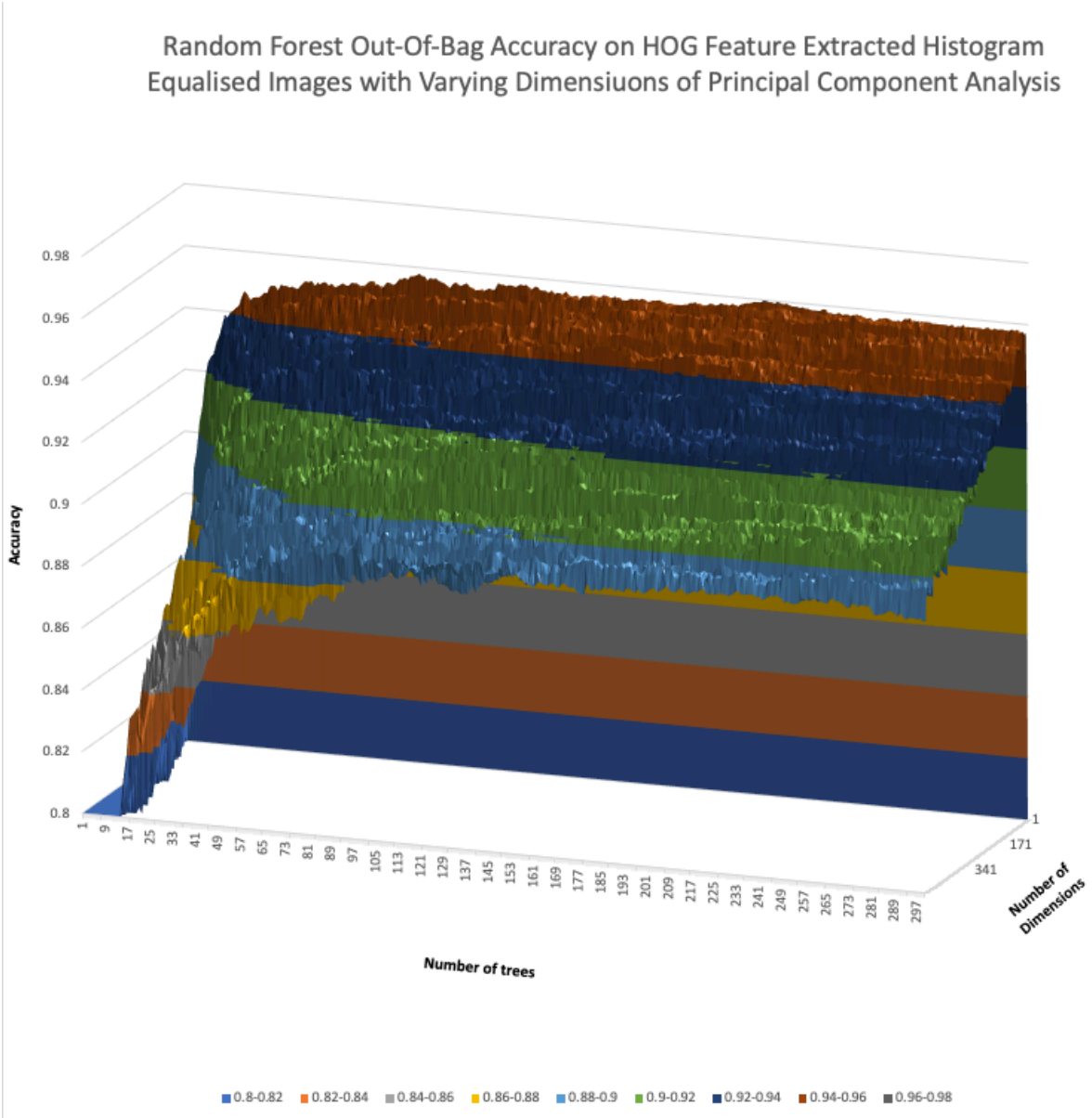


Figure 3.1: Analysis on the Number of Trees and Number of Dimensions of a Random Forest Under Principal Component Analysis

From Fig 3.1 z-axis, it can be seen that as you increase the number of dimensions, the accuracy of the random forest model rises rapidly until a point, in this model, 11 dimensions.

From the x-axis, as what is consistent with the random forest models outlined previously in this report, the accuracy of the model rapidly increases as you increase the trees until a certain threshold where there is an insignificant effect on increasing the number of trees to increase the accuracy of the model. Referring back to Fig 2.1 and 2.2, it is clear that reducing the number of dimensions reduces the number of trees required. On the highest accuracy number of dimensions, most of the accuracy increases come from increasing the number of trees from 1 to 50.

In model, the parameters that yield the maximum accuracy of 0.960 using **11 dimensions** and **212 trees**.

In order to access the accuracy of this model in relation to the accuracy of the SVM and KNN models, a 5-fold accuracy test was taken. The accuracy of the test was **0.951**, a slight 1% improvement over the non-PCA model of 0.942.

The forest was trained using 300 trees on Histogram Equalized images that had gone through LDA with varying proportions of the PCA dimensions being used in the LDA analysis.

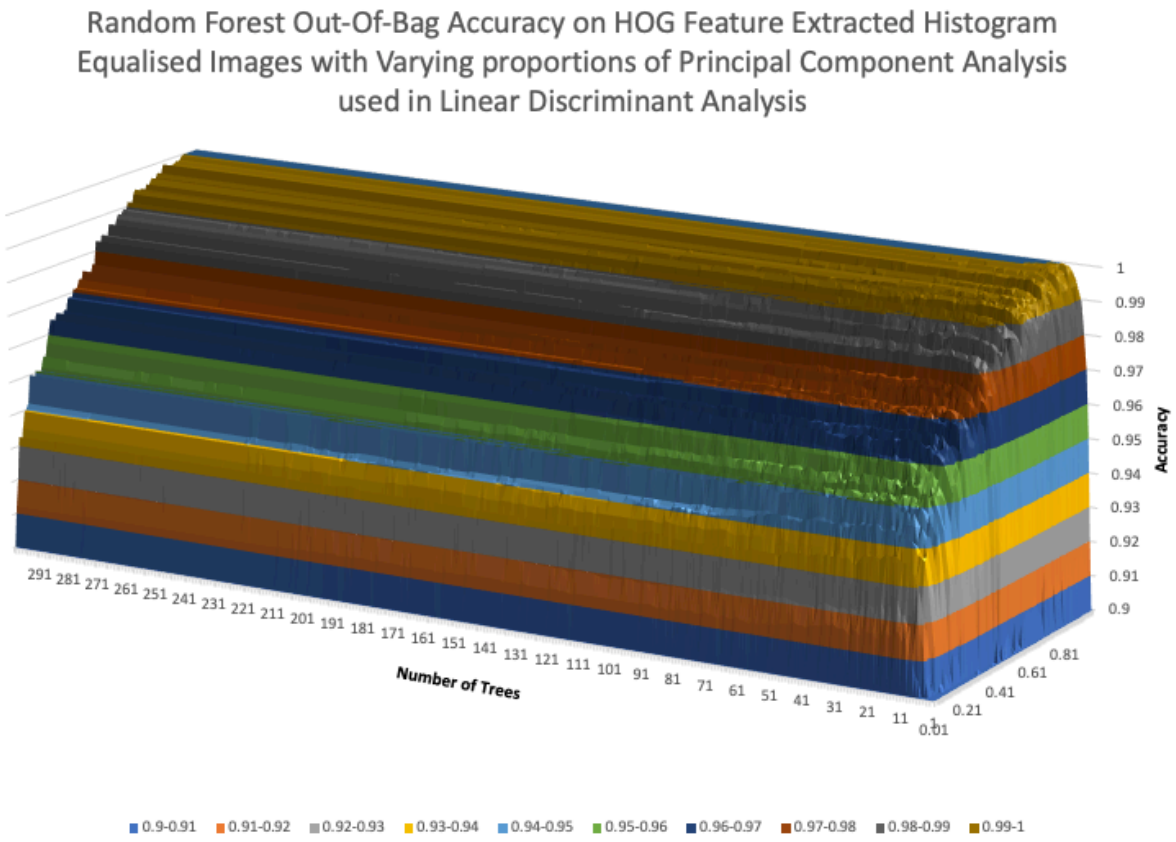


Figure 3.2: Analysis on the Number of Trees and Number of Dimensions of a Random Forest Under Linear Discriminant using different proportions of Principal Component Analysis

From Fig 3.2, looking at the x-axis, it is clear that increasing the number of trees needed beyond 35 had little to no impact on the accuracy of the model. This fits in with the pattern that the less dimensions are being analyzed, the less trees are needed.

The peak out-of-bag accuracy of 1 came from using 57 trees and 98% of the most important PCA dimensions.

In order to access the accuracy of this model in relation to the accuracy of the SVM and KNN models, a 5-fold accuracy test was taken. The accuracy of the test was **0.902**, less than the accuracy taken for PCA or non-dimensionally reduced images.

SVM – PCA

The SVM accuracy was determined by using a 0.2 holdout on the 3000 test images. 5-fold cross validation was too computationally expensive for the computer being used to run the analysis. A 0.2 hold-out gives an estimation of what the 5-fold cross validation reading would be.

As was described earlier in the report, the SVM automatically selects the ideal parameters. See below the PCA results:

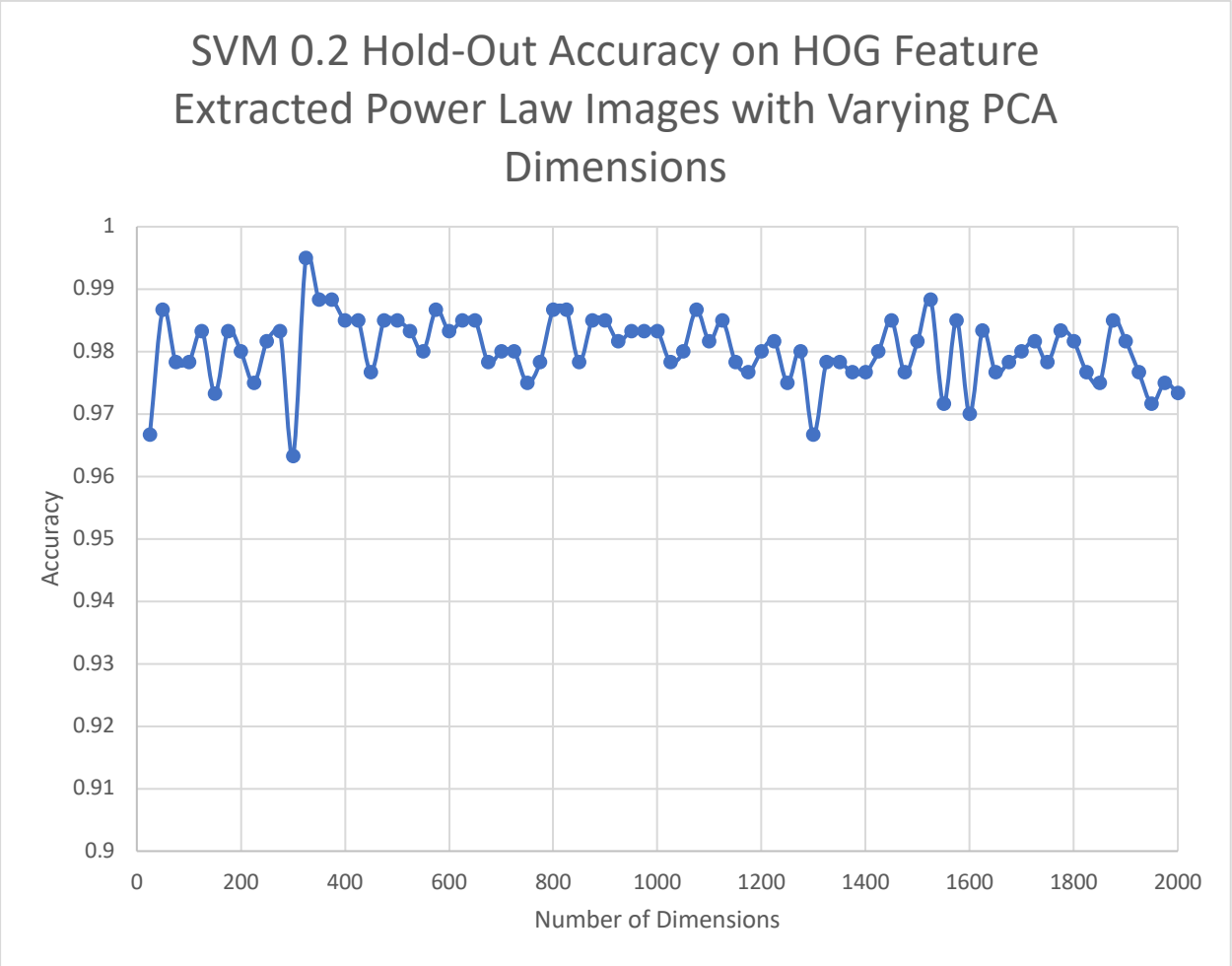


Figure 3.3: SVM Hold-Out Accuracy on HOG Feature Extracted Power Law Images with Varying PCA Dimensions

From Fig 3.3, there is a lot of variation in the results. This is due to the fact that the results were obtained by using hold out instead of 5-fold. Therefore, the results will be less reliable.

The maximum accuracy of 0.995 was achieved by SVM using 325 dimensions.

Due to the fact that LDA will separate the images over 1 dimension, the computational cost was reduced and thus 5-fold cross validation was performed:

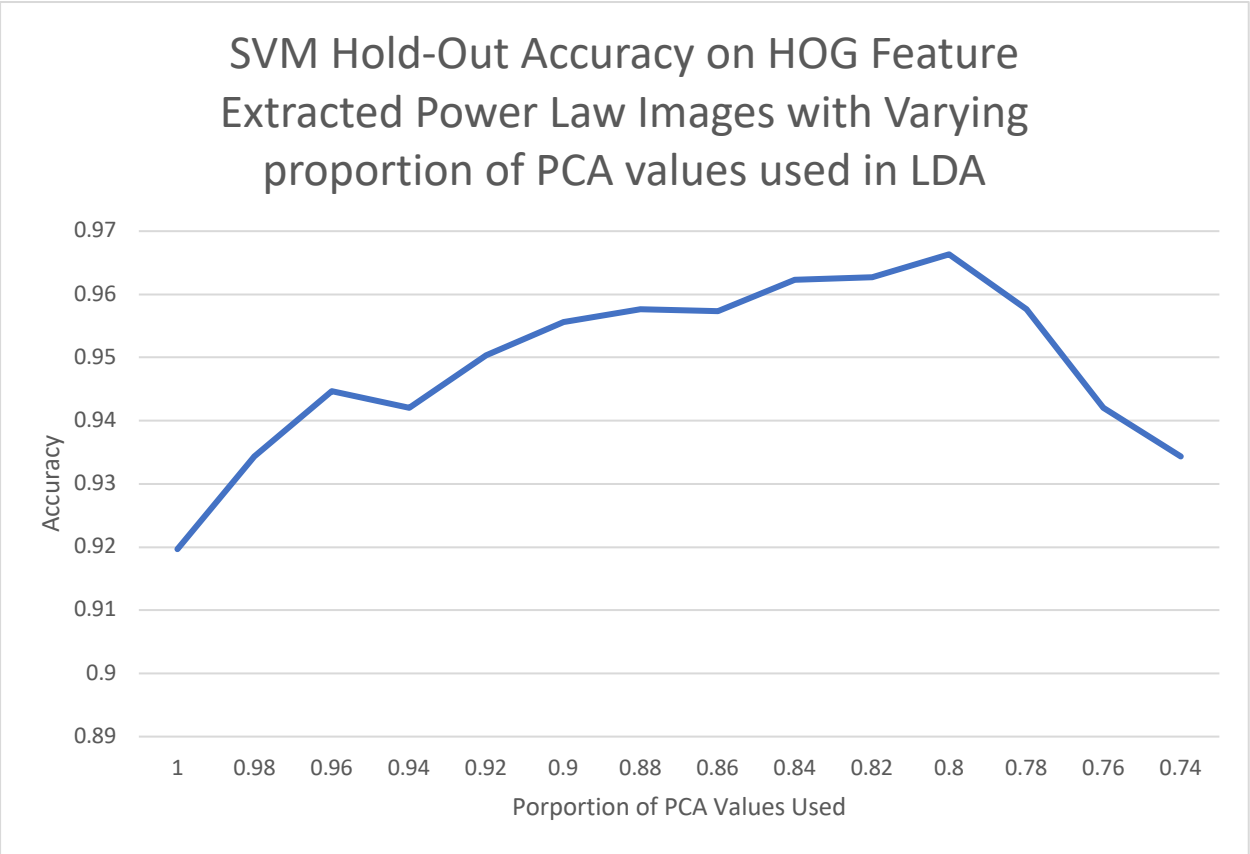


Figure 3.4: SVM 5-Fold Cross Validation Accuracy on HOG Feature Extracted Power Law Images with Varying PCA Dimensions used in LDA

The maximum accuracy produces was **0.967** using 0.8 of all the PCA dimensions. The peak accuracy of this model was less than the peak accuracy of the PCA model before. In fact, LDA did not improve over the standard non-dimensionally reduced model which had an accuracy of **0.975**.

Time to compute

In order to test the time taken to train and test the models, a 0.2 hold out was used to train and test the models. See the results below:

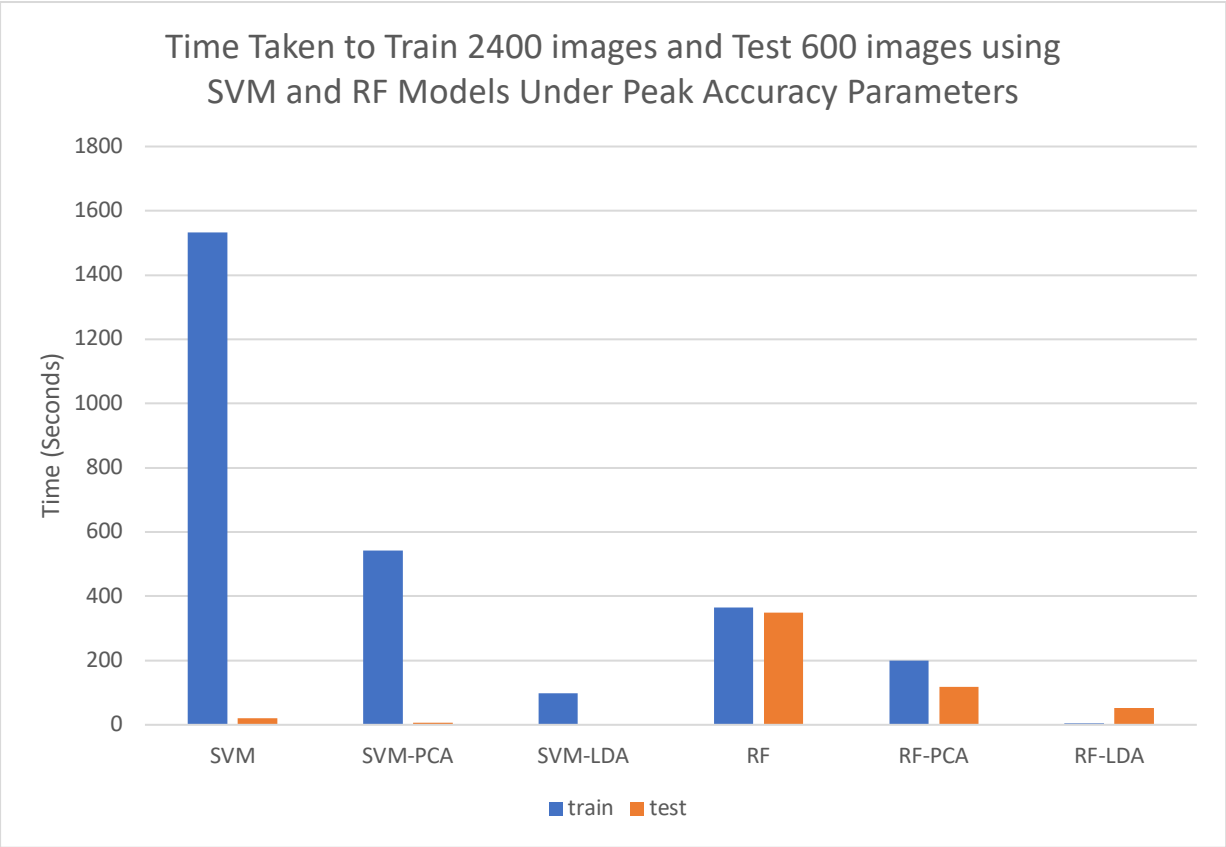


Figure 3.5: Time Taken to Train and Test SVM and RF Models Under Peak Accuracy Parameters

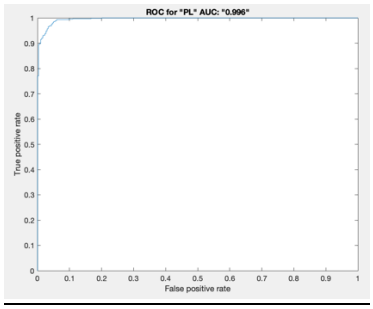
While reducing the time to train is helpful, the real time advantage of dimensionality reduction comes from the time to test the images. It will allow for quicker video processing.

As can be seen from Fig 3.5, SVM is considerably quicker to test images compared to random forest.

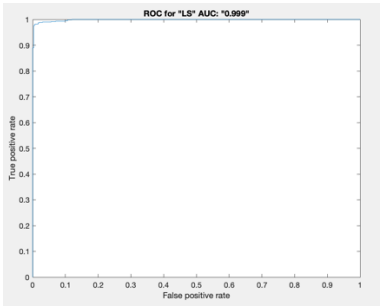
ROC graphs for Models

To help figure out which model will be the best, ROC graphs were created in order to try and visual show which models provided the best true positive rates to false positive rates. These graphs were created using the Testing Datasets provided by the partition_hold_out function. We are looking for the highest Area under Curve(AUC). The higher the AUC, the better the model is at predicting the desired class.

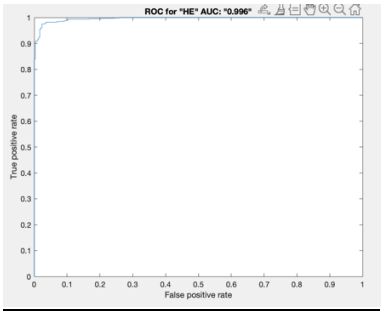
To create the graphs the function showROC was created. This function takes in the test dataset and the model. It also takes in a string, but this is only to display the model type on the outputted graph. The test set is the individual classed using the SVM detection function provided to us. This function has been modified slightly to return the probability of the prediction being a 1, not just the highest probability. This is necessary for the creation of the ROC graph. The built in Matlab function "perfcurve " is then used to get all the necessary data needed to plot the ROC graph, it returns the x values, those being the false positives, the y values, which are the true positive rates, and the AUC. The x and y values are then plotted into the graphs shown below.



PL model with AUC of 0.996



LS model with AUC of 0.999



HS model with AUC of 0.996

As these figures show. The AUC all are very similar, only being around 0.003% different than each other. That number does also vary depending on the test dataset. This is due to the fact that the test dataset is randomly generated whenever the svm_driver is run. Testing was run several times to try and get an average of AUC for each. The average for PL was 0.997, the average for HE was 0.997 as well and the average for LS was 0.998. these numbers were so similar that picking one above the other didn't seem relevant.

While the ROC graphs give a good visual description on the overall reliability of the model, in this case there was not enough differences between the models to determine an outright superior classifier

Chosen Methods

The KNN learning method was discounted before any pre, processing techniques. It can be seen from Fig 1.1 and 1.2 that the accuracy of the model was considerably less than that of SVM or random forest.

From Fig 3.3, The quickest and most accurate model is SVM using power law images with features extracted by HOG and dimensions reduced to 325. However, for reason stated earlier, the results are less reliable than others due to the lack of 5-fold cross validation. Therefore, the same model without the dimensionality reduction will also be used to analyze the video and the results compared.

Post-testing Evaluation of Hypotheses

The following hypothesis analysis will be taken in relation to the chosen model above. The power law pre-processing technique yielded the most accurate results, thus H1 is accepted and H0 and H2 is rejected. In accordance with the machine learning hypothesis, SVM turned out to be the best model so H0 was accepted while H1 and H2 were rejected.

Sliding Window

Sliding window was chosen to be the means of detection for this project. As the camera was also moving along with the pedestrians, it was deemed not viable to use any form of key frame detection. Sliding window fit the purpose of these frames as it can be set up to create multiple windows of different sizes. In doing this, it would allow the detector to hopefully find the pedestrians close to the camera and further away from it. A basic overview of the process behind the sliding window implemented within this project. The sliding window goes across the image at different steps and with different sized windows. The windows are then converted into a HOG feature and that feature is then passed into the SVM classifier to be checked to see if the classifier detects a person in the image. The pre-processing of the images is done at the loading stage and was set to Power law as that had been determined to be the best fit for SVM and HOG during training and testing.

The first parameter of the sliding window that needed to be calculated was the ratio of the window itself. The images used for the classifier have a height of 160 and a width of 96. This is a ratio of 1.66. This was a good place to start as it meant that the resizing of the images would have the same ratio as those of the training and testing images. This was calculated using an X scale. This scale had values set to a 1:1.66 ratio. The pedestrians image was then divided by these numbers to give a value for the width and height based on these numbers, the height of the pedestrian images being 480 and the width being 640. For example, if the value was 1. The width was being divided by 1.66, giving a value of 386 and the height was being divided by 1, giving a value of 480. This was would be the size of the first scale window. Due to the size of the pedestrian image, this created a window size with the incorrect ratio, around 1.24. The window was resized to 160 x 96 in order for the classifier to be applied to it, however the windows were too square and so the classifier was finding a lot of false positives. As shown in the figure below:

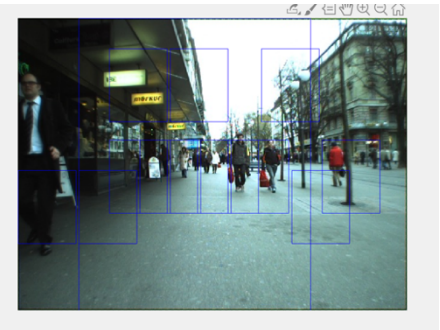


fig 1.(1.66 ratio boxes)

This was deemed too vague and so a new way of getting the window size was needed. Looking at the results from test.dataset, it showed that the results windows from it were at a ratio of around 1:3, fir first object has a width of 43 and height of 128, giving a ratio of 2.98.This made more sense as the pedestrians were roughly 3 times taller than they were wide. This gave the result as shown in the following figure:

fig 2.(ratio 1:3 boxes)

The red labels show the transformed score returned from the SVM classifier. The boxes also seemed to contain a lot more than just the pedestrian. This lead to the creation of the get_boundry_box_size() function. This function reduces the bounding box of the object by a quarter. Below are the before and After.

Before resizing ***After resizing***

Fig 3.(boundary comparison)

Now that the window ratio had been finalised. The next stage was to determine the step size the windows would take. The code provided had the sliding window go across the image in a grid pattern. Jumping the width and height of the window at every step. This caused massive areas of the image to not be checked correctly and did not give accurate classifications on pedestrians. As seen below.

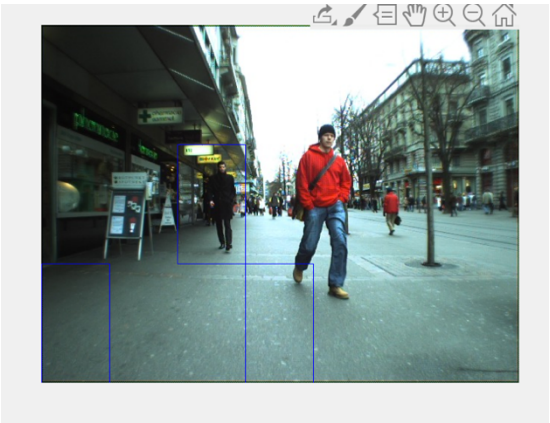


Fig 4. (grid windows)

The first logical number to use was 1. This would mean that the window would move across each image 1 pixel at a time. Ensure that nothing was missed. While this would be the most effective way of checking every part of the image, due to the size of the image and the number of pixels. This would mean that there would be over 175,000 images checked. That was only for 1 window size. The sliding window needed to be run with multiple sizes. It wasn't even possible to get a time for how long this would take. It was then decided to look at numbers that fit within the boundaries of the image. That is, number that divide perfectly into 480 and 640. It was determined that the smaller numbers, 1,2,4,8 and 16 where too small for steps and would have taken far too long to process each image. The next number tried was 40. While this did give ok results, it was still felt like the steps were too big. This meant that the window, only passed across the image 16 times per row. The number 20 was finally used to check how accurate this was, and the results below show the success of having the step size as 20. This also struck a good balance of timing to accuracy as it only took around 20 seconds to do a pass of the whole image using a single window size.

Once the window ratio and steps had been determined, the amount of different sized windows needed to be worked out. At first there was 9 different ratios of windows. [1.25, 1.5, 2, 2.5, 3,3.5, 4, 5, 6]. This causes problems because it was taking over 5 mins to run on a single image and the results that were coming back gave the smaller windows random false positives. As seen below. These were being selected by the NMS function due to their higher confidence score.

Fig 5.(small false positives)

The scale at which the windows would generate was eventually fixed to between 1 and 5, with intervals of 0.1. this was done by leveraging the parallel toolkit available for Matlab. This allowed

multiple cores to run at once and so cut down the time needed to run all the different window scales. It was able to run all scales in around 400 seconds with 4 cores active.

Another parameter that was chosen was the cut off for the score given by the SVM classifier. The classifier returns either a 1 or a 0 if it determined that that there was a person in the window it was detecting. the SVM classifier also returned a value of confidence on which prediction it believed was the correct one. After transformation, this number was between 0 and 1. This meant that different detected objects had different levels of detection. This lead to problems with the classifier identifying trees and signs as people.

Fig 6.(false positive on tree)

To correct this a secondary check was added to the prediction. Once the SVM had predicted a 1. The score was checked. If the score was below a threshold then it was changed to a 0. The final threshold decided was 0.85. This number was selected as it allowed for most of the trees and sign detections to be eradicated, without losing any of the pedestrian detections.

These parameters, the window scale, the step and the threshold were set. The sliding window could not go to work trying to identify the pedestrians in the image. A problem soon arose of overlapping and very similar boxes surrounding the same object within the image. The next step in the process was to apply non maxima suppression to the image so that only the most confident boundary boxes show up around the detected pedestrians.

Non maxima suppression (NMS)

This is the function of reducing the overlapping boundary boxes that are present whenever multiple sliding window detect the same thing. Without NMS, the result looks like the figure on the left below, However once it is applied, we get the result on the right.

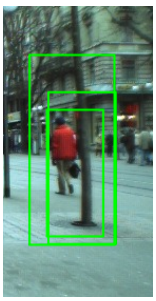


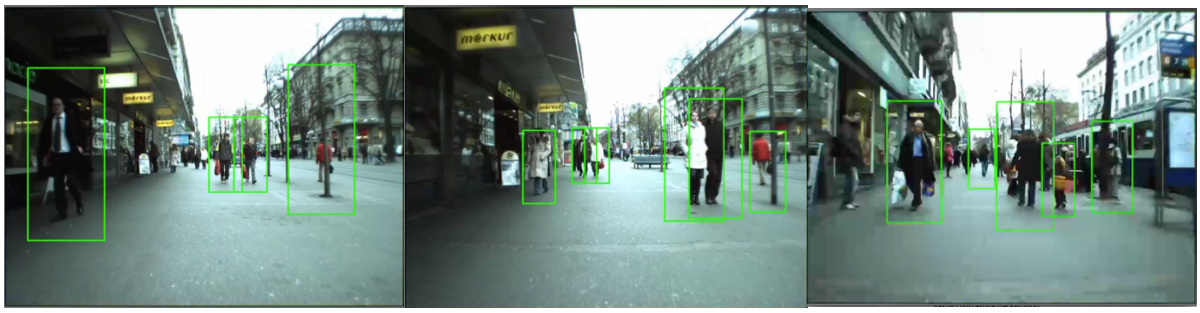
Fig 7.(pre NMS)

fig8.(post NMS)

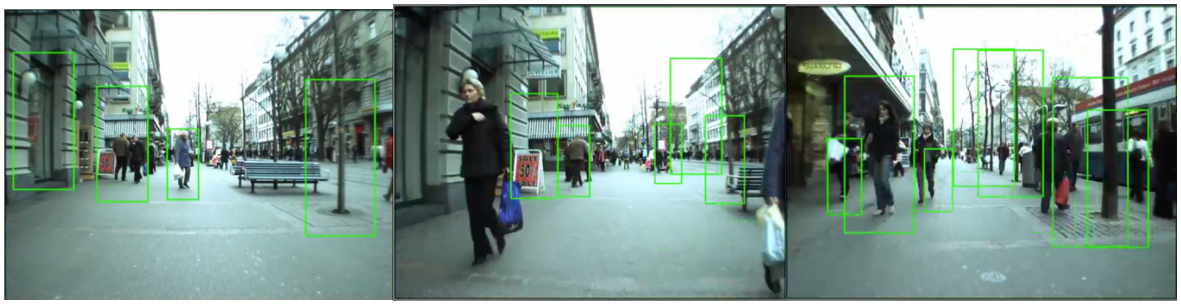
This is done by checking every objects boundary box with each of the others in the image. then the intersection of these boundary boxes was checked to see if they were above a given threshold. If the threshold was met, the box with the higher score was kept, and the lower score was removed. Through trail and error it was determined that the correct threshold was 0.8. any more than that and people walking beside each other were losing boundary boxes. Less than that and the false positives like legs were being chosen more than the whole person.

Visual Analysis of Detection Results

Overall, based on a visual analysis of the final video, the decisions made on the sliding Window and the optimized SVM classifier allowed for a fairly accurate pedestrian detector. The examples below show that there are certain frames that have 100% true positives in the image. While it hasn't detected everyone. It has detected the majority of discernable pedestrians in the frame. The threshold for NMS as also allowed for pedestrians walking together to be classed as 2 individuals as can be seen below.



While some frames were very accurate with their classifications, others were not. The same problems appeared to rise again and again, with false positives such as legs, trees and signs being confidently and incorrectly classified. This could be down to the fact that those objects have very similar features to that of a pedestrian. That being, an object in the middle of the image with a vastly contrasting background.



As the images shows. The tree on the right and the shop window are both classed as pedestrians. It has managed to correctly identify the people in the middle but the contrast between the window, tree and their backgrounds shows that further work would need to be carried out to remove these false positives. The third image also shows that legs were detected at a higher confidence to the rest of a person. Causing that boundary box to be selected.

While it may not be possible to solve these problems with the current classifier, there are measures that could be taken in trying to limit the number of false positives in a frame. The number of steps could be increased. A drop from 20 pixels to a smaller number, or have different windows sizes. These changes could allow for the classifier to miss the trees and legs entirely or give the people a more confident classification which would allow them to be selected over the lower valued objects.

Quantitative Analysis of Detection Results vs Ground Truth

The F-measure was going to be the deciding factor to determine whether the dimensionality reduction played a positive role on detecting a person in a video. The overall F-measure of the dimensionally reduced model had aa F-score of 0.6134, compared to 0.6322 for the non-dimensionally reduced model.

The main difference for this was caused by an increased rate of false negatives. It seems that the dimensionality reduction removed too much detail to the point where people were not getting identified.

Over the full set of 100 pedestrian images the following results were achieved from the non-reduced SVM model-

Overall Accuracy	0.6881
Recall	0.5702
Precision	0.7093
F Measure	0.6322

Power Law – HOG – SVM (Using all 3000 images for training for maximum accuracy)

Scale	1:0.1:5	A ***large sliding window scale of 1 to 5 in 0.1 increments*** was chosen to compliment the high score cut off. The reasoning for this is that we discovered ***a broad scale with small steps leads to more opportunity to have highly confident classifications*** which meet the score cut off of 0.85.
Step	20	We experimented with several step sizes. However, we needed one which was small enough and also ***a factor of the dimensions of the pedestrian images***. A step size of 20 is perfect in terms of dimensions and smallness – however this coupled with the broad scale is ***computationally intensive***.
Score Cut Off	0.85	The score cut off is the ***minimum confidence we allowed for a classified window to be considered a pedestrian***. It is important to note that we applied a score transform of 'logit' for 0-1. The problem domain is finding a fine balance between false positives and false negatives. ***If the score cut off is too low, there are more false positives. If the score cut off is too high, there are more false negatives.*** To defeat this problem, we opted for ***a broad scale with a small step to ensure every opportunity for highly confident classifications*** which can meet this strict cut off. This succeeded mostly but due to the computational power required was limited somewhat. To increase accuracy, we would even further increase the scale range, and decrease the step size – this would lead to more opportunities for highly confident classifications.
Workers	8	To be able to cope with the computational intensity of a broad scale and a small step size – ***we utilised parallelism***. This allows each scale to be performed on a different thread on the CPU. To squeeze as much power as possible, we increased the default maximum workers in the ***Parallel Computing Toolbox*** configuration from 4 (1 for each core) to 8 (1 for each thread). This places the CPU under 100% load but is the fastest way to compute in this case.

C	37.894	This was the optimal C value returned from the Bayesian Optimisation performed earlier.
Sigma	3.465	This was the optimal sigma value returned from the Bayesian Optimisation performed earlier.

This was a computationally intensive task, if you are going to run this on anything other than a high-power desktop CPU, reduce workers to number of cores, reduce the scale range and increase the step size. Took exactly 3 hours 21 mins to complete on Intel Core i7 6700k (4 Cores, 8 Threads).

We are happy with the F Measure of 0.6322 and have developed a fairly competent classifier and sliding window which can scale upwards in terms of accuracies given more computational power. However, there is only so much the model can do – and would benefit from extra features i.e. face.

The code for comparing our results with those in the dataset folder – to do the initial comparison it requires the sliding window driver to have been run first to generate 'total_objects' in the environment.

dataset_driver.m | import_test_dataset.m | get_comparative_me

```
clc;
output_metrics = get_comparative_metrics(total_objects);
save output_metrics output_metrics;
```

However the overall accuracy, TP, FP, and FN for the optimized model were saved in models/optimised_baseline and we convert these to the evaluation metrics using get_eval_metrics stored in the same folder.

get_eval_metrics.m | SVMTraining.m | README.md | showROC.m | sh

```
clc;

% The below was created by running 'dataset_driver' in dataset

load('optimised_baseline_metrics.mat');

% col 1 = accuracy per image
mean_accuracy = mean(output_metrics(:,1));
% col 2 = TP
TP = sum(output_metrics(:, 2));
% col 3 = FP
FP = sum(output_metrics(:,3));
% col 4 = FN
FN = sum(output_metrics(:,4));

recall = TP / (TP + FN);
precision = TP / (TP + FP);
% specificity = TN / (TN + FP);
f_measure = 2 * TP / (2 * TP + FN + FP);
% false_alarm_rate = 1 - specificity;

% Too cumbersome to calculate true negatives and unknown quantity.
```

We did not gather the true negatives, as this would be an incredibly large number given the broad sliding window scale we have combined with small steps. As a byproduct of this we were unable to calculate specificity and false alarm rate.

Conclusion

Overall, we created a competent classifier which is configurable in a large number of ways. This was a learning experience for us all – and that is why we experimented with so many different combinations – to understand fully the significance each part of the process has to the final accuracy/evaluation metrics.

If we were to repeat the project, we feel an additional feature would be highly beneficial. For example – facial recognition perhaps using Haar-like features.

As far as our solution goes – the only way accuracy can increase is with an ever-broader scale and a smaller number of steps on the x and y axis the sliding window takes. This is not feasible in terms of computational cost.