CSC3067: Machine Learning and Video Analytics
Group Project assignment

Thomas Reid - 40263793
Alexander Wilson - 40294480
Mathew Holmes - 40292229

08/12/2022

1 Training

1.1 - Full image

This project makes use of several different feature descriptors. The first descriptor that was investigated within the project was the use of full image as a feature descriptor.
The first step was to convert the image (which was stored as a 2D image) into a 1D matrix consisting of all the data. The use of a full image feature descriptor is a great way to capture and describe the features of an image. A full image feature descriptor allows for a comprehensive representation of the features in the image, which can be used to identify objects within the image. This type of descriptor is generally more robust to changes in the image, such as rotation, scale, and lighting, which can help improve the accuracy of the algorithms in various conditions.

1.2 - Hog

This project also makes use of the Histogram of Oriented Gradients (HOG) descriptor to capture the features of the images. Before training the system, the Hog features for each image were calculated which resulted in a 1D matrix with 72 values. Each of these matrices are then stored inside a 2D matrix which can then be passed to the learning method. HOG is an effective and widely used feature descriptor that has been used successfully in many computer vision applications. HOG is a powerful descriptor that captures local shape information by dividing the image into small blocks and computing the distribution of the gradient directions within each block. HOG is particularly well suited for object recognition tasks, as it can accurately capture the edges and shapes of objects within the image. HOG is also a computationally efficient descriptor, which makes it a great choice for this project as it allows us to accurately capture the features of the images and use them to identify objects within the image.

1.3 - Gabor

In addition to HOG, Gabor features have been used as a feature descriptor. Similarly to Hog, before passing data to the learning method each image's gabor features are calculated as a 1D matrix before storing the features for each image in the dataset within a 2D matrix. Gabor features are a type of filter-based feature descriptor that is used to capture texture information from images. Gabor features are well suited for texture analysis, as they are able to capture the local frequency and orientation information of the image. Gabor features are also able to capture the subtle differences in textures between different objects, which can be useful for object recognition tasks. Therefore, Gabor features are an excellent choice for this project, as they can help to accurately capture the texture information of the images, which can be used to identify objects within the image.
The main difficulty with using Hog features is that for each training image, it generated a 1D matrix containing 19440 different values, this results in several downsides, it

significantly increases the computational cost of processing each image, and whenever you try to differentiate objects with a large amount of dimensions, it creates large areas of empty space, reducing the accuracy of the classification. One way to get around this problem however is to use Dimensionality reduction techniques.

1.4 - Dimensionality reduction techniques

Dimensionality reduction techniques are able to transform large datasets into smaller ones that still contain the majority of the information while dropping the least important variables. By reducing the dimensionality of the data, the model can focus on the most relevant features, thereby improving the accuracy of the model. Additionally, using dimensionality reduction techniques can help to reduce the computational cost associated with training the model. In this way, Gabor feature extraction combined with dimensionality reduction techniques can help to improve the performance of the model.

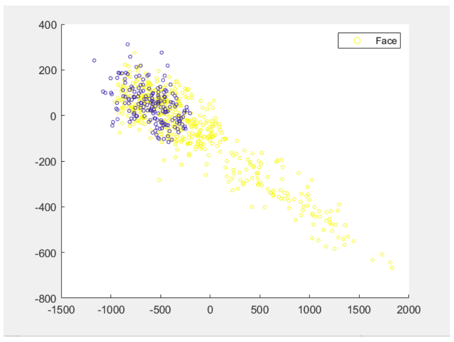
There are two main types of dimensionality reduction techniques; PCA (Principal Component Analysis) and LDA (Linear Discriminant Analysis). PCA is an unsupervised algorithm which is used to reduce the number of features in a dataset while retaining most of the information. It looks for correlations among features and produces a set of new features that explain most of the variance in the dataset. LDA is a supervised algorithm which is used to reduce the number of features in a dataset while maximising the separation between different classes of data. It looks for patterns in the data that can be used to distinguish between different classes and produces a set of new features that are most useful for distinguishing between different classes.

Within the project, LDA has been used for dimensionality reduction since LDA is better at preserving class-discriminative information. This means that the features selected by LDA will be better at distinguishing between different classes of objects in images, while PCA may select features that are more generic or less class-specific.

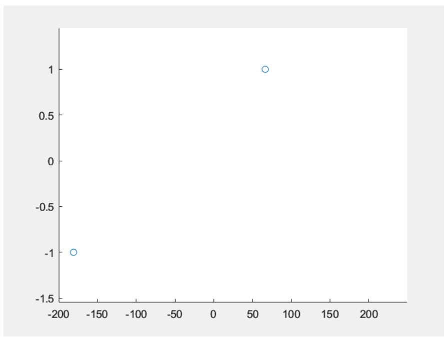
This is implemented within the project by passing all the training images and labels into the function LDA(), which returns an array containing data such as the eigenvectors, eigenvalue and the mean of the dataset. The variable xLDA is returned which gets set as the image data, as it now contains only one value for each image. You can easily notice the separation between the face and non-face images as the values are 66.471 (Face) and -180.95 (Non-Face). Included below is a graph, displaying the output from PCA vs the output from LDA.

To test the model using gabor data we need to convert the 1 x 19440 matrix into a single value. There is a line of code that uses the mean and eigenvectors calculated from the training dataset and creates the appropriate testing data. This will convert it to a single value based on the previous transformation which will then allow the learning method to output a classification for the image.

PCA output



LDA output



1.5 - Preprocessing

One of the most important pre-processing techniques used on both training and testing data is contrast enhancement. Contrast enhancement is used to increase the contrast between different objects in the images, making them easier to identify and recognize. This can help to improve the accuracy of the models used in computer vision. By increasing the contrast between different objects, the model will be able to more accurately identify and recognize them.
Contrast enhancement is implemented on both the testing and training dataset, as well as within the sliding window detection, with the aim of making any faces easier to detect. The reasoning behind using contrast enhancement as the only preprocessing technique is due to it being the only technique that would make a significant positive difference on the dataset. Since the training images are so small, there is very little noise within the images. If the image got passed through a noise reduction filter, it would blur the image and reduce the quality of the output. Contrast enhancement will also brighten the image so there is no need to use an additional brightness enhancer on the testing and training data.

2 Learning methods used

2.1 - Nearest Neighbour

The first learning method used was Nearest Neighbour. Nearest neighbour learning method is used in computer vision models because it is a fast and efficient way to classify images. By using this method, a computer can quickly compare a given image to a set of already known images to determine the best match. Additionally, it is a non-parametric method, meaning it does not require any parameters to be set to work.
To use the nearest neighbour learning method, each image is individually compared with each training sample, calculating the euclidean distance between each testing sample's feature value and the corresponding training sample's feature vector. Whichever training sample's euclidean distance sum is the smallest, it will classify the testing sample as the label for the respective training sample.

2.2 - K Nearest Neighbour

The KNN algorithm is almost identical to nearest neighbour however it works by finding the K most similar training samples to the given input image and then takes the label of the majority of those samples as the output label. The KNN algorithm is also robust to outliers, meaning it can work well in noisy datasets. Using the K-Nearest Neighbors algorithm for computer vision models is an efficient and accurate approach that can provide accurate results with minimal training data.

2.3 - SVM

The last learning method explored within the project was Support Vector Machines (SVMs). SVMs are an effective machine learning algorithm used for predictive modelling and classification of data. They are particularly effective because they use a mathematical optimization technique called the kernel trick to find a hyperplane of optimal separation between the two classes. This approach allows for a higher degree of accuracy in classifying the data than other methods. In addition, this method also allows for a more robust decision boundary, which makes it more robust to noise and outliers. Overall, SVMs are highly accurate, robust to noise and outliers, and can be used to perform complex non-linear classification tasks. Therefore, they are a great choice for computer vision models.

3 Testing

3.1 - Testing and training data

To ensure that consistent testing and training data is used throughout the project, Each file makes use of the datasets, face_test.cdataset and face_train.cdataset. These two files contain both positive and negative examples with their respective labels classifying them as either a 1 (face image) or -1 (non-face image). Using these datasets means that when using different types of feature extractions and learning methods, the data within the system is consistent, allowing the models to reproduce the same evaluation metrics each time since the model will be trained and tested with the same data each time.

3.2 - Cross validation

Ten fold cross validation is a useful technique used to evaluate computer vision models. It is a robust method that provides a reliable estimate of model performance by splitting the available data into ten distinct subsets and using each subset as a test set in turn. This technique will provide a better understanding of a model's performance in a more accurate manner than a single evaluation, as it provides a more comprehensive evaluation of the model's ability to generalise. Additionally, the use of ten fold cross validation helps to guard against overfitting, as the model is not exposed to the same data multiple times.

4 Evaluation

4.1 - Metrics

To analyse the performance of the different models, there was a range of different evaluation metrics used. The first metric investigated was the model’s recognition rate. The recognition rate is an effective metric to evaluate a computer vision model as it shows the accuracy of the model in recognizing objects in an image. It provides a numerical value that can be compared between different models, allowing for easy comparison of their performance. Additionally, recognition rate is a good indicator of the model's ability to generalise to unseen data. By using recognition rate to evaluate a computer vision model, It will provide a better understanding of its performance and determine its suitability for various tasks.

Only looking at recognition rate can be deceiving however. To get a better understanding of each models' performances it is advised to calculate a confusion matrix. A confusion matrix is a useful tool to analyse the recognition rate of a model by comparing the model's predicted labels to the true labels of the data. It gives a detailed breakdown of the performance of the model and can be used to identify where the model is making mistakes. By using a confusion matrix to evaluate a computer vision model, it allows for a better understanding of its performance and determines its suitability for various tasks.

We can use these values from the confusion matrix to calculate even more in-depth metrics, which will provide a better understanding of how the model performs in certain situations. The first metric derived from these values is the model’s precision. Precision is an important metric to evaluate a computer vision model because it measures the model's ability to accurately classify objects in a given image. It helps identify false positives so that the model can be improved. By using precision to evaluate a model,it will give a better analysis of how the model is performing at its best and identify areas for improvement.

The next metric derived from the confusion matrix is called the recall value. It measures how accurately a model can identify all relevant instances of an object in an image, even when these instances are difficult to detect. A high recall value will indicate that the model is well suited for classifying instances of an object by using computer vision.

Another metric that can be investigated is the model’s specificity value. It measures the ability of a model to correctly identify negative samples, which can help determine if the model is prone to false positives. Specificity is also useful in determining if the model is able to distinguish between different classes, as low specificity can indicate that the model is failing to distinguish between classes. By taking specificity into account, it is possible to gain a better understanding of how well the detection models are performing.

The last evaluation metric calculated is the F1 Score. The F1 score is an ideal evaluator as it is a balanced metric that takes both precision and recall into account. The F1 score is a measure of a model's accuracy and its ability to correctly identify positive examples, and the higher the F1 score, the better the model is performing. Additionally, the F1 score is less sensitive to small changes in the number of true positives or true negatives, which makes it more reliable than other metrics.

4.2 - Models results

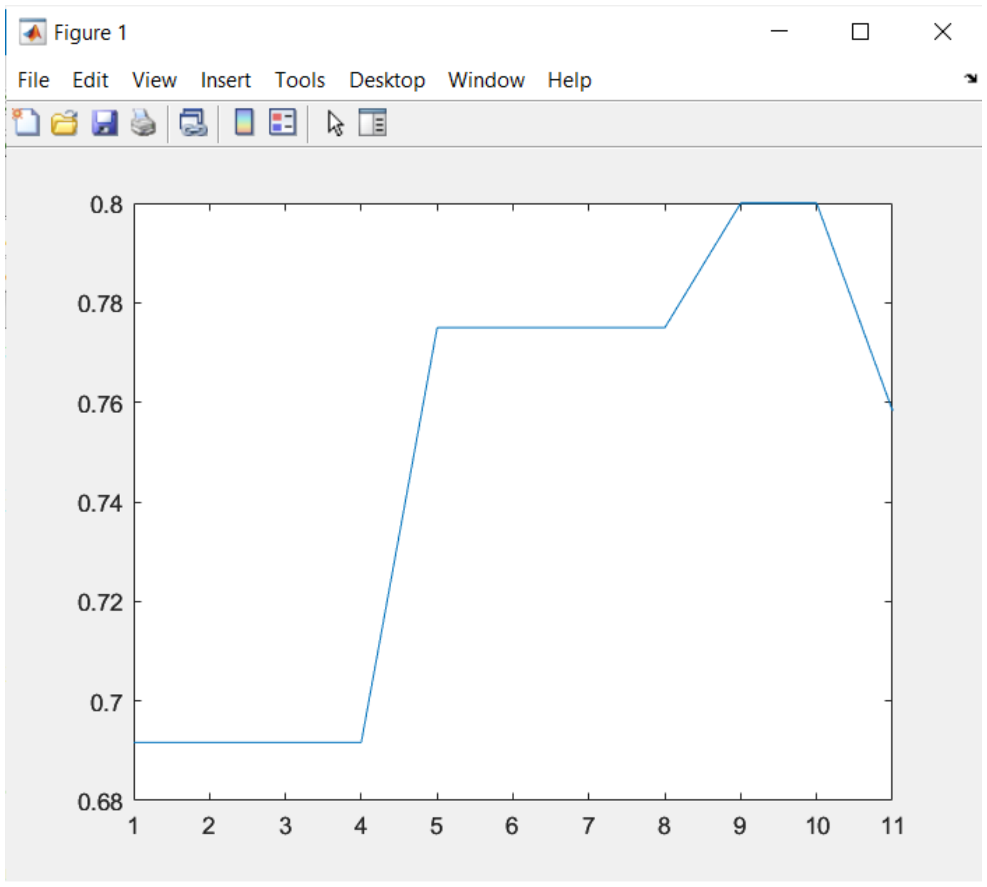
Below you can see the Evaluation metrics for each model, using different types of feature extractors. This is using the data stored in the face_train and face_test datasets.
NN Results

Feature Extractor	Accuracy	TP	TN	FP	FN	precision	specificity	recall	F1 score
Full Image	0.692	150	16	24	50	0.862	0.4	0.75	0.802

Hog	0.746	165	14	26	35	0.864	0.35	0.825	0.844
Gabor	0.904	184	33	7	16	0.963	0.825	0.92	0.941

KNN Results

When using KNN as a learning method, it will work differently depending on the value of K, since the model will output different predictions. Below is a graph displaying how the accuracy changes for different values of K.. As you can see when K = 9 we get the best accuracy value, 0.8. The accuracy will fall off after 9 because we start to overfit the model on the given training data which results in a worse accuracy when testing against new data. This is how well the model performs when K = 9.



KNN Results for K = 9

Feature Extractor	Accuracy	TP	TN	FP	FN	precision	specificity	recall	F1 score
Full Image	0.775	170	16	24	30	0.876	0.4	0.85	0.863
Hog	0.846	188	15	25	12	0.883	0.375	0.94	0.910
Gabor	0.904	184	33	7	16	0.963	0.825	0.92	0.941

SVM Results

Feature Extractor	Accuracy	TP	TN	FP	FN	precision	specificity	recall	F1 score
Full Image	0.833	200	0	40	0	0.833	0	2	0.909
Hog	0.786	155	34	6	45	0.963	0.85	0.775	0.859
Gabor	0.904	184	33	7	16	0.963	0.825	0.92	0.941

4.3 - Results explanation

There is lots of interesting information that can be investigated from these analysis tables. Initially, looking at the performance of the NN model it is possible to see that its performance wasn't overly impressive. When using a Full image feature extractor it was able to correctly identify just under 70% of samples. It was better at identifying the positive examples of faces as it had a precision of 86%. The model performs slightly better when using Hog, however through using the images Gabor features, with their dimensions reduced using LDA, the accuracy significantly increased to be able to correctly identify 90% of the samples

Through changing the model from NN to KNN, it is possible to identify some interesting results. KNN seems to be a better identifier when using full image, having an accuracy of 77.5%, this may have increased due to KNN's better resilience to outliers within a dataset. It also performed significantly better when using HoG features too, with nearly every metric increasing. Through further analysis you can conclude from the model's true negative and false negative examples that it has improved at recognising instances where no face is present in an image. Another interesting observation that can be seen from using KNN is that it performed identically to NN in every metric when using a dimensionally reduced gabor feature vector. This will most likely be due to LDA's ability to maximise class distinction whenever it reduces the size of the vector, and given that it is the same reduced data testing the system for both NN and KNN, it means that similar outputs will be given when run through different classifiers. Looking further you can see that this is also the case with SVM when using a dimensionally reduced gabor feature vector.

An interesting observation we can see when looking at the SVM results is that the use of a full image feature extraction will only return us with positive results. This is not intentional. Each image that gets tested will return the same value, which therefore provides no distinction when trying to differentiate between classes. I believe this is due to an error within the code.
Looking at the differences of the tables as a whole, each model is able to differentiate between face images and non-face images with some level of confidence. Overall the NN model works worst. This may be down to how simple the algorithm is, as well as its

liability to outliers within the data. Especially when using full image feature data, since there are no distinctive features taken from the testing and training images that would help the model when it comes to classification.

To test the true abilities of the models on unseen data, a sliding window detection system will need to be implemented, which can scan an untested image and return all the instances that it recognises a face. We will use SVM as the model for this implementation since SVM is powerful at detecting complex patterns within data. This will allow for experimentation with different feature extraction techniques to see what method performs best.

5 Implementation

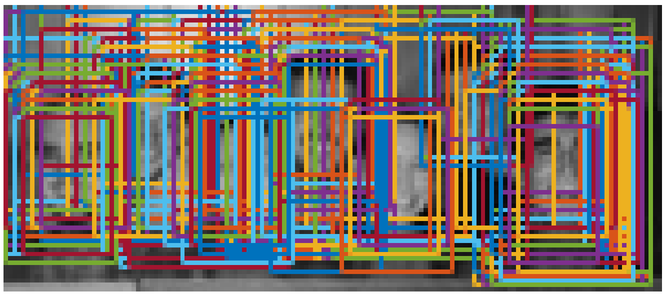
5.1 - Sliding window

To begin with implementing a facial detection application, the first step was to create a sliding window object detection algorithm. It is a type of local search algorithm that divides an image into a grid of overlapping regions and then computes a score for the regions based on their similarity to a given target object. The primary benefit of using a sliding window detection technique is that it allows for a large amount of flexibility. This is because the window can be adjusted to fit any shape or size to identify the object in the image. Additionally, the window can be moved to different locations within the image, allowing for different views or perspectives of the same object. This allows for more accurate object detection.

The code which was provided in practical 5 was used as a template for the sliding window object detection. First, the detector calculates the coordinates of everywhere within an image that the window will fit. Next, the image is divided into small sections, called windows where each window is evaluated for the presence of a face. If a face is detected, the window is marked and the search continues. The window size and scale will be pre-calculated before the sliding window detection begins. Once it has tested every position for the presence of a face, it will change the size of the window and try again, continuing this cycle for a pre-allocated amount of iterations.

However, the code from practical 5 had different requirements than what is necessary for creating a facial detection system, and the code was altered in several ways. Since the size of the window is changing upon each iteration, this makes it hard to compare the selected area to any of the training images, mainly due to the training images being smaller in size. One thing that had to be done was to ensure the window can be reshaped into an area the same size as the training images, which was 27 x 18 pixels. This means, if the model is using Gabor or Hog images, it will have to calculate them from a scaled down version of the window image.

Once the sliding window detection has run through every position of the image at every iteration, it will output a matrix that contains all the possible positions a face could appear, the height and width of that window and the confidence that it has identified a face. This will give the program all it needs to be able to display a bounding box over the identified face. However this creates another problem that will need to be solved, since the result of the sliding window detection will contain many overlapping bounding boxes.



Direct output a sliding window without NMS applied

5.2 - NMS

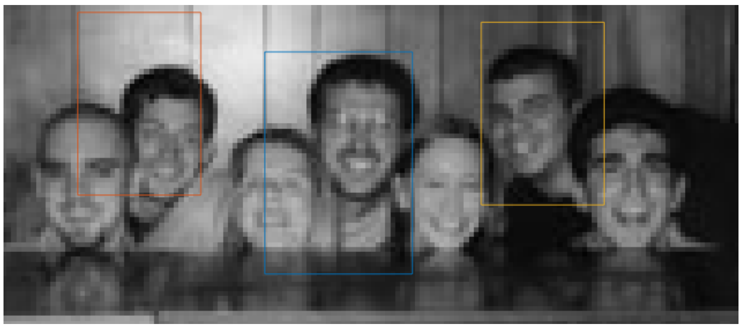
Non maxima suppression is an important step in the sliding window detection process for identifying faces. Without this step, the detection process may generate multiple overlapping detections for the same object, resulting in an overly cluttered, inaccurate result. Non maxima suppression works by analysing the output of the detection process, which consists of a series of bounding boxes, and removing any detections that overlap significantly with each other. By removing the overlapping detections, the output of the detection process is more accurate and visually less cluttered.

5.3 - Results

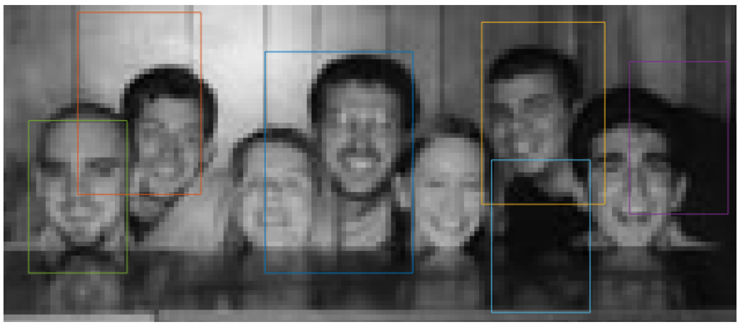
Included below are the output images for each model, along with the threshold value that was passed through to the NMS function. There were four images to test the implementation of the model's on. Using different threshold values when using non-maxima suppression will allow us to see different variations of results
When using full image, the classifier is identifying every image as a positive value, because of this, it will not work as an effective classifier, so therefore the detectors will use SVM as the learning method and shall experiment with using Hog and gabor features as the main feature extractors.

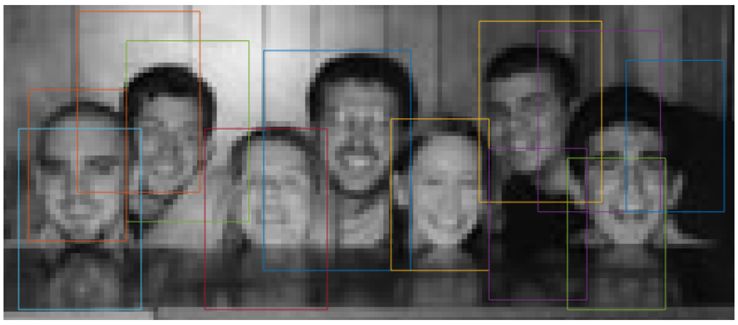
SVM HoG Results

Test Image 1:

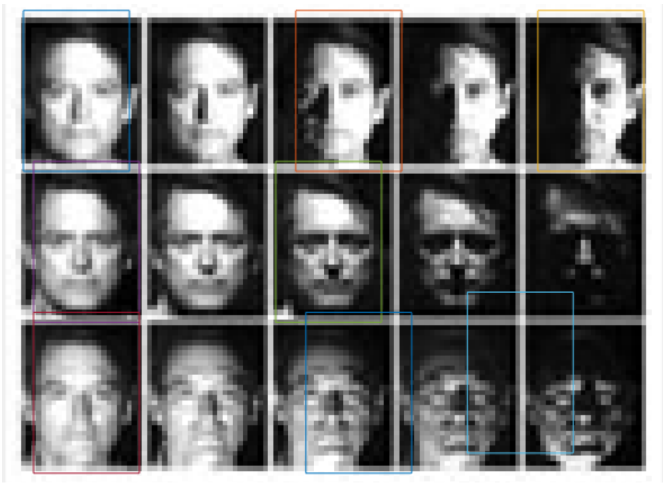


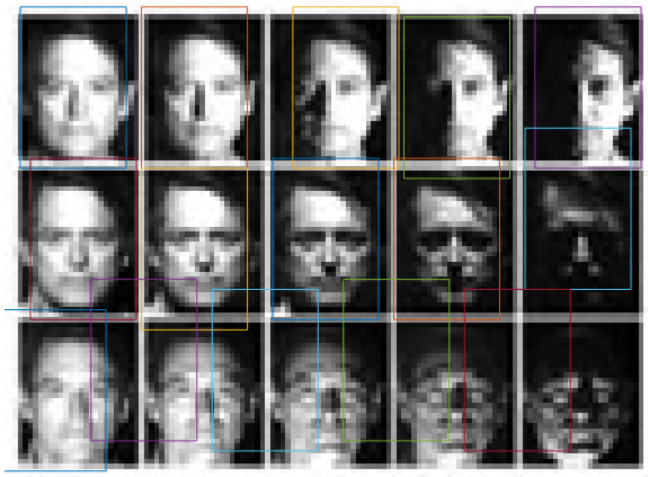
Threshold = 0.1


Threshold = 0.5

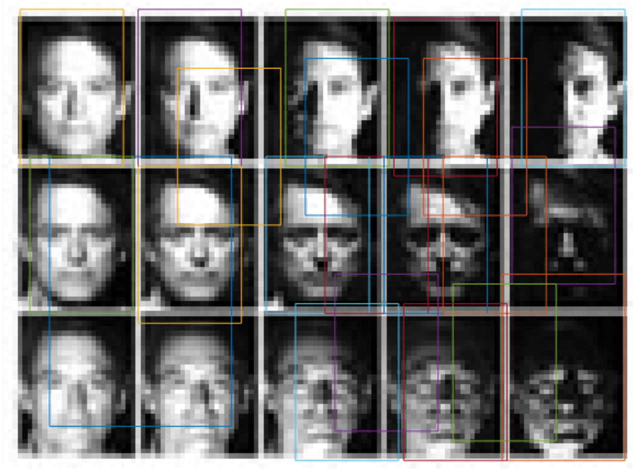

Threshold = 1

Test Image 2:


Threshold = 0.1

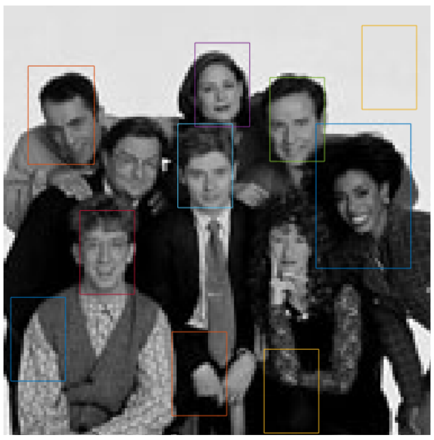


Threshold = 0.6

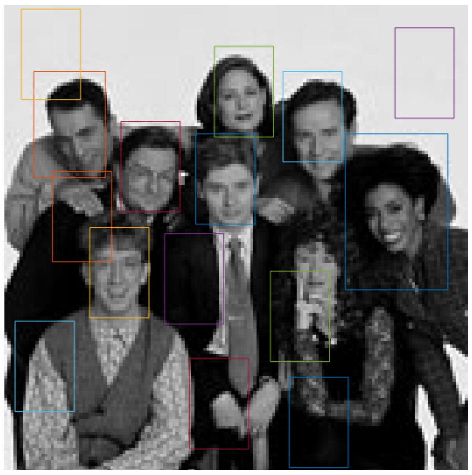


Threshold = 1

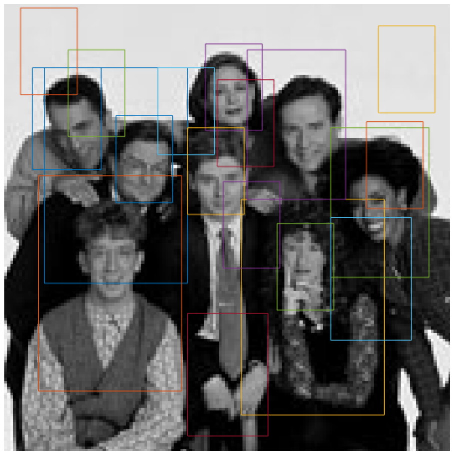
Test Image 3:



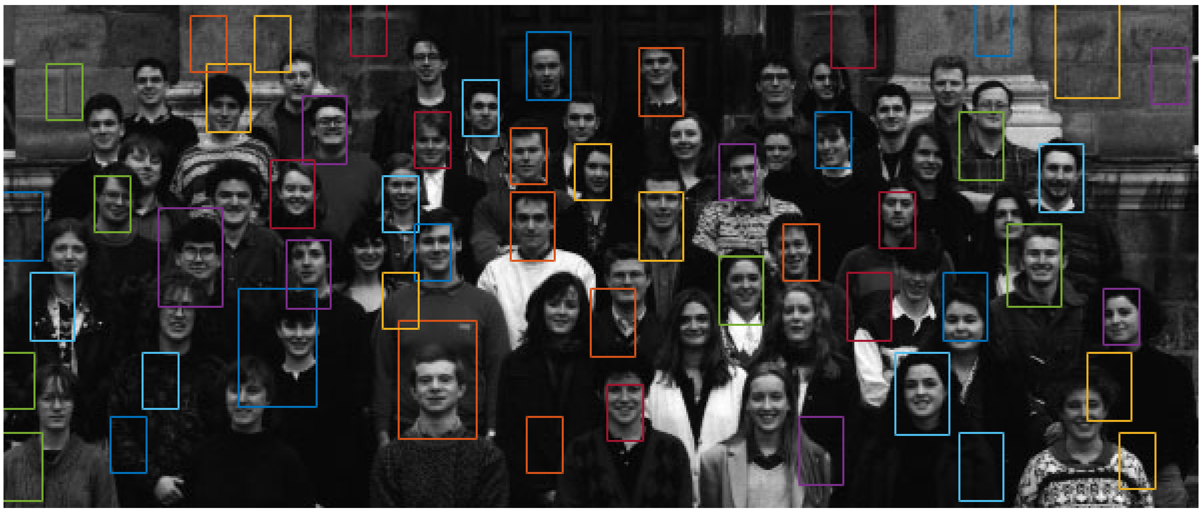
Threshold = 0.1

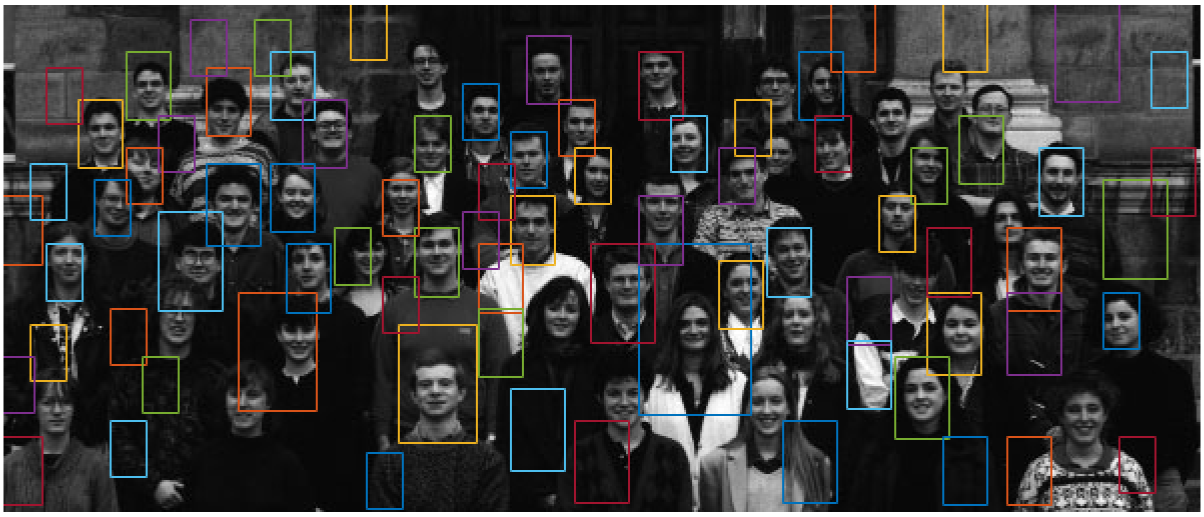


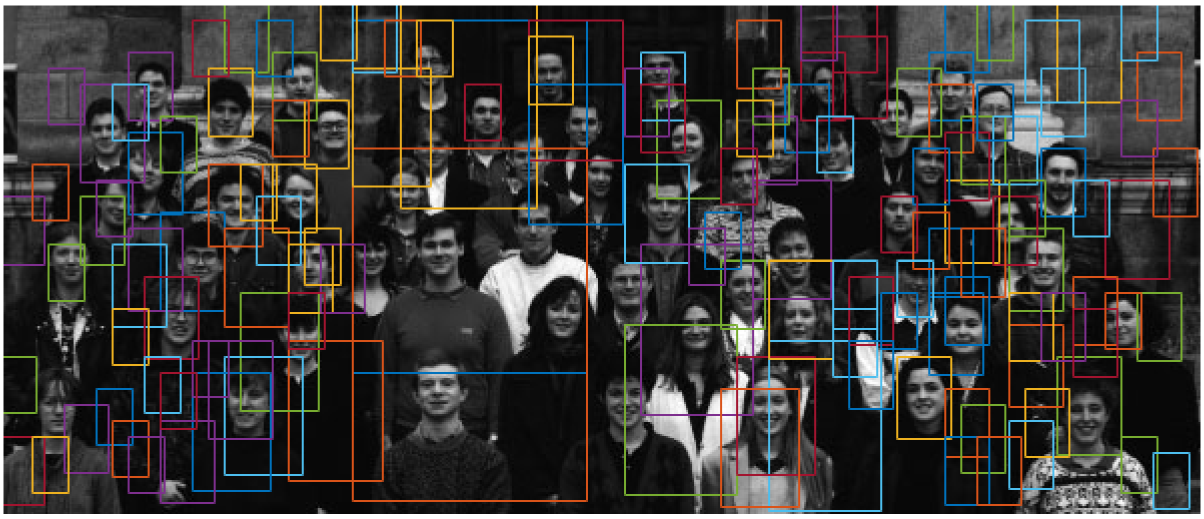
Threshold = 0.4


Threshold = 1

Test Image 4:

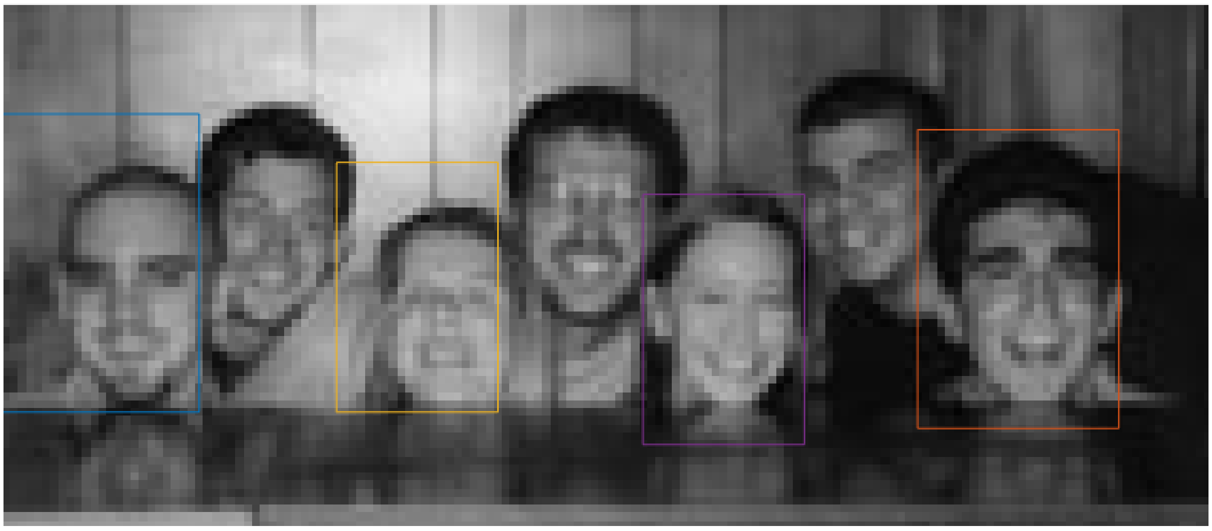

Threshold = 0.1

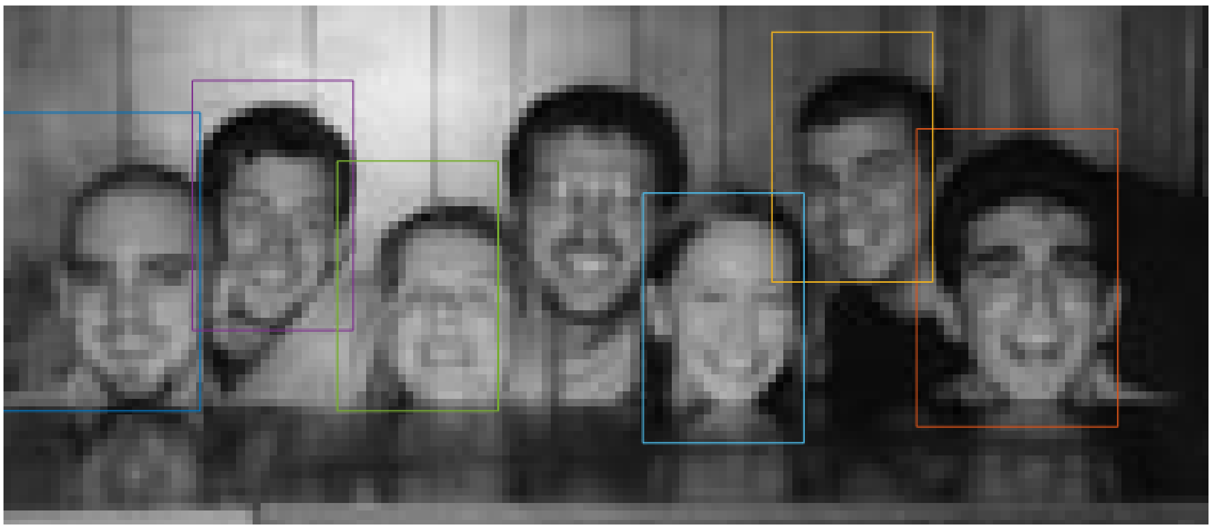

Threshold = 0.4


Threshold = 1

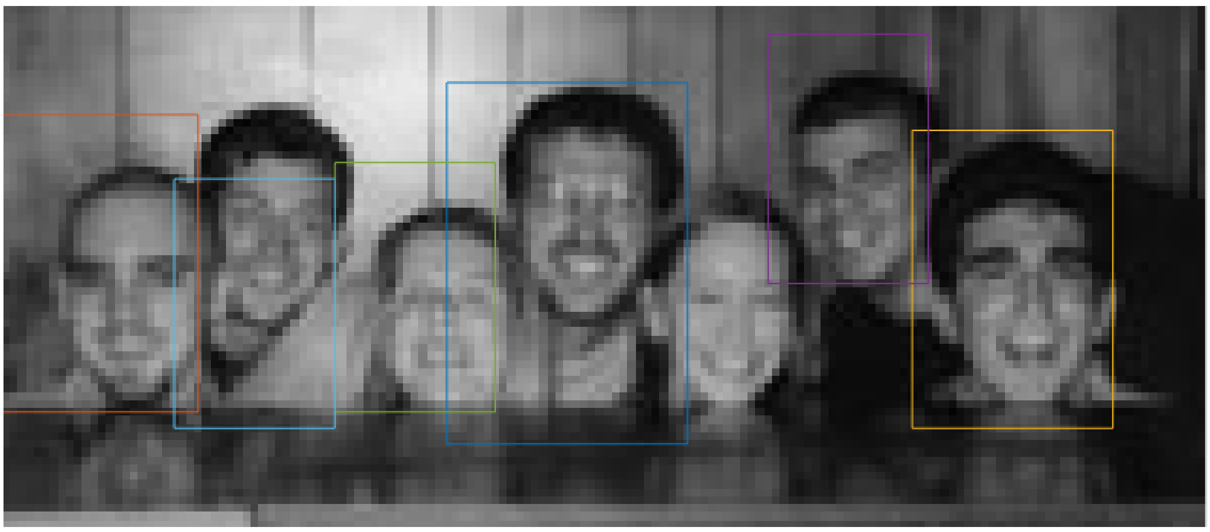
SVM Gabor Results

Test Image 1:


Threshold = 0.1

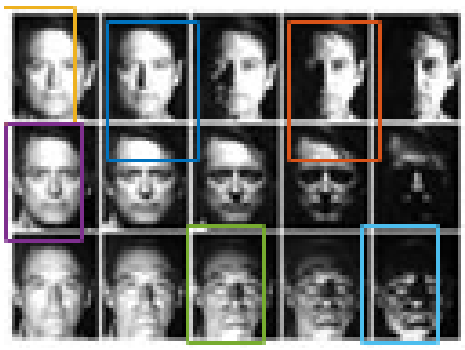


Threshold = 0.7

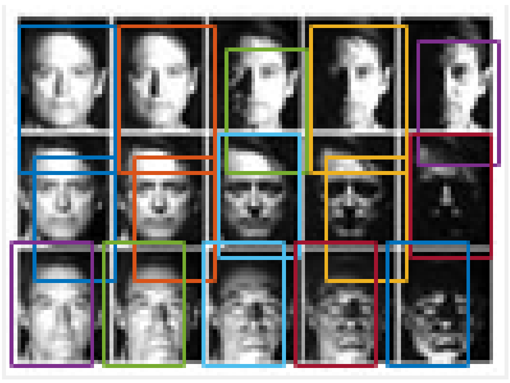


Threshold = 1

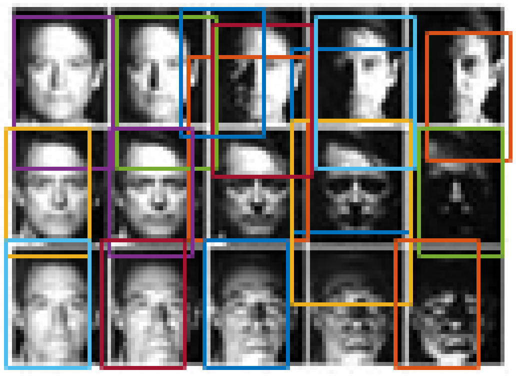
Test Image 2:



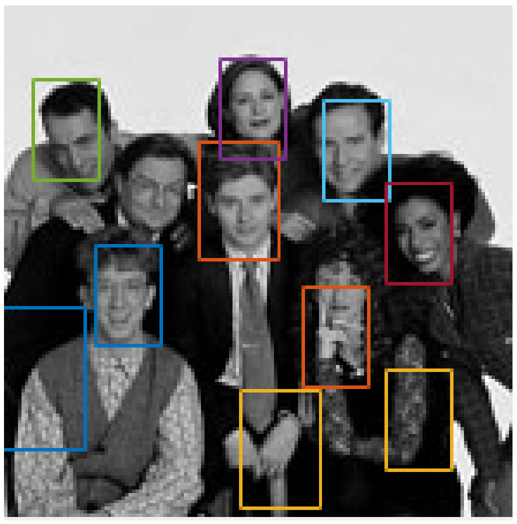
Threshold = 0.1

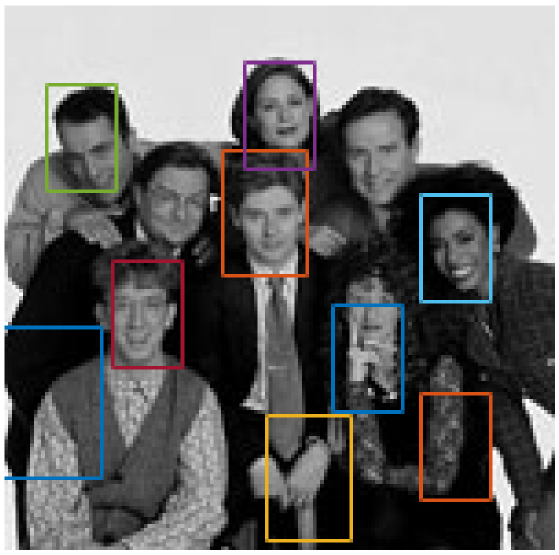


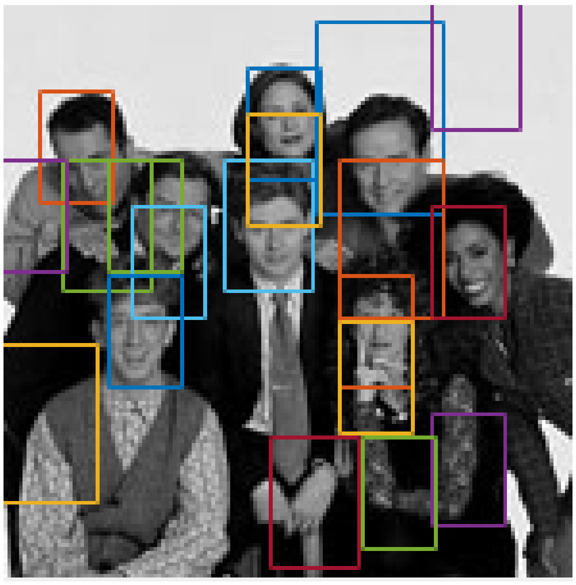
Threshold = 0.5


Threshold = 1

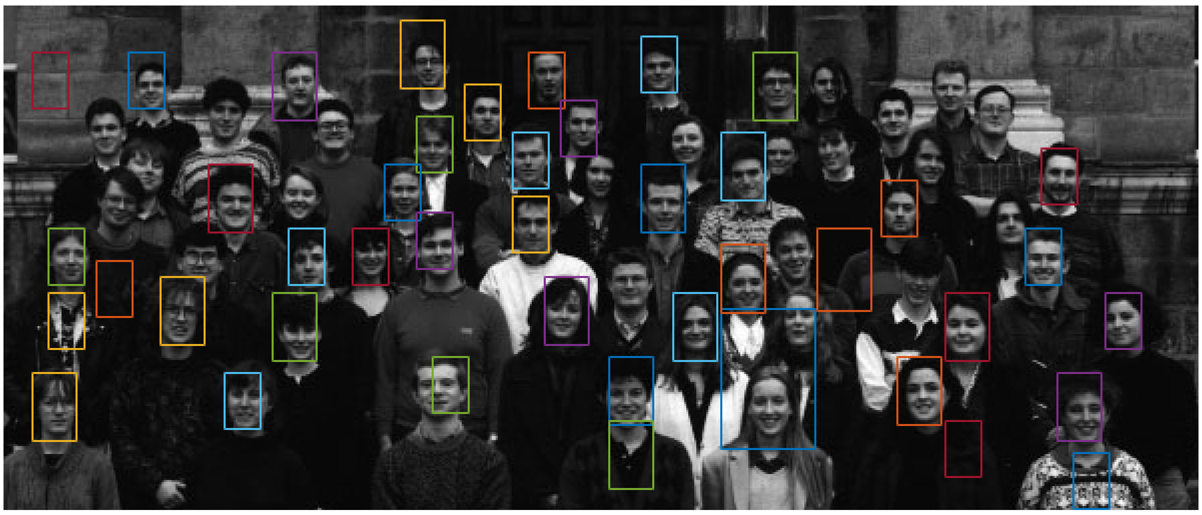
Test Image 3:

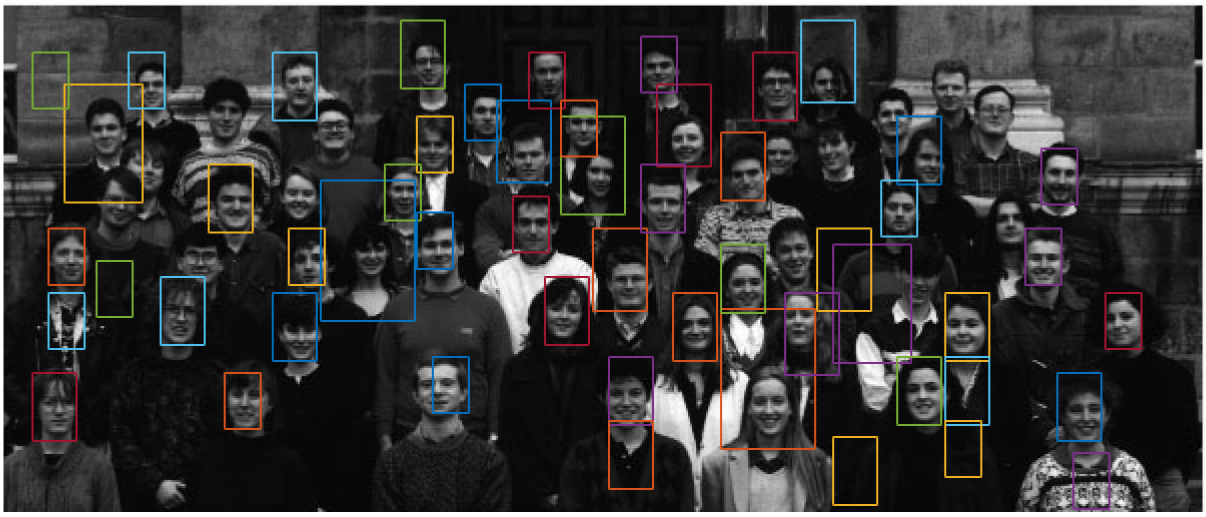

Threshold = 0.1

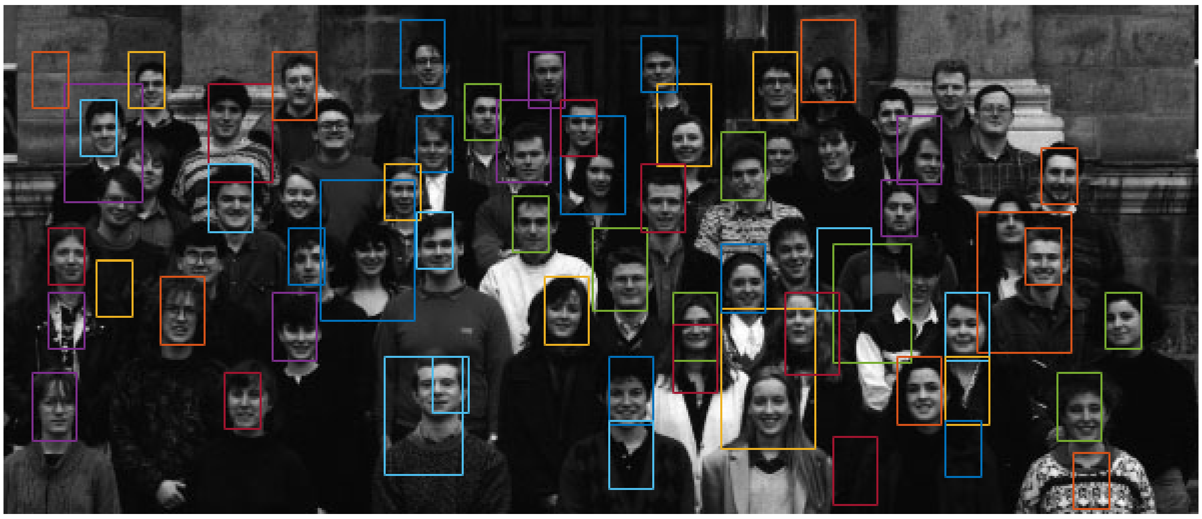

Threshold = 0.2


Threshold = 1

Test Image 4:


Threshold = 0.1


Threshold = 0.9


Threshold = 1

5.1 - Implementation results analysis

To be able to investigate the accuracy of these models, it is best to track as many metrics as possible, to be able to perform a better analysis of how the detectors work in real life circumstances. Upon investigation of how the threshold affects the output of the detector, the image was tested at extreme values, as well as its optimal performance. It will be the optimal performance that the evaluation metrics will be calculated from. To calculate these metrics, the number of TP, FP and FN values were manually counted by hand, which means there may be a small margin of error.

SVM HoG detector

Test imager	TP	FP	FN	precision	recall	F1 score
Im1	4	2	2	0.667	0.667	0.667
Im2	11	4	4	0.733	0.733	0.733
Im3	6	4	2	0.6	0.75	0.67
Im4	40	36	17	0.526	0.701	0.601

SVM Gabor detector

Test imager	TP	FP	FN	precision	recall	F1 score
Im1	6	0	1	1	0.857	0.923
Im2	15	0	0	1	1	1
Im3	5	4	2	0.556	0.71	0.634
Im4	43	9	16	0.827	0.729	0.775

There are many things that can be observed from the results of both implementation detectors. When using Hog features as the main feature extractor, it seems to work well at identifying individual instances of faces, for example, when testing on im4.jpg, it correctly identified 40 of the 59 faces within the photo. However it is not without faults, the model seems prone to detecting false positives.
However when using Gabor features the results were significantly more impressive. It reached a maximum precision score on two occasions, and correctly identified all instances of faces without making one false classification, when testing the second image. Since each image is a different size, The models may not work as effective at different scales which is something that was investigated. Both performed better when working with smaller images, and struggled with image 3, this might be due to the distribution of faces. Within image 3 there are a lot of faces in close proximity, and it is very possible that the non-maxima suppression has cancelled out a bounding box around someone's head because it overlapped with another bounding box. It is also possible that the way the faces are presented within image three, having different rotations and angles, that the model struggled to classify them.

6 Conclusion

Looking back on how each system performed throughout, at different points it is not possible to say outright that one model is superior to another. Each different type of learning method, and feature extraction will have pro's and con's that will make it suitable for different applications. If you were to look at the results of how each learning method performed, it might be easy to say that SVM was the best performing model, but SVM is a slower process than KNN and NN. If it was required that the model work in real-time, these methods may be better suited.

There are several things that could improve our models, if more time was allocated. It would be interesting to see how a full-image detector would have worked with sliding image. Another interesting idea would have been to use LDA within the HoG feature extractor, since it still contains high dimensional data.

There are also many other functions and learning methods that could have improved the output of the system, for example, it would have been interesting to see how a neural network would perform against these simpler learning methods.