Automated Facial Detection Using Machine Learning

AUTHORS

Matteo Melis - 40324932
Samuel Agnew - 40325924
Matthew Stewart - 40332822

2024-12-05

Contents

1 Introduction

In this paper, we present our findings from deploying several machine learning models to detect upright, frontal views of faces in grayscale images. To address this challenge, we explore a range of classifiers with the goal of identifying the optimal approach for the task and developing a robust final detection system. Our investigation includes traditional algorithms such as Nearest Neighbours, K-Nearest Neighbours (KNN), and Logistic Regression, as well as more advanced techniques like Support Vector Machines (SVM) and Random Forest. We hypothesise that advanced models will outperform simpler models, as SVM is known to perform well with small datasets and non-linear decision boundaries. Furthermore, ensemble methods such as random forests, can be particularly effective at identifying "non-face" images, a challenging aspect of face detection. We also explore the use of a neural network, however, we predict this to perform rather poorly due to the limitations of our dataset.

Training a machine learning model for face detection presents unique challenges, primarily due to the difficulty in characterising prototypical "non-face" images. Unlike face recognition, where the goal is to distinguish between different faces, face detection involves discriminating between "images containing faces" and "images not containing faces." While it is relatively straightforward to obtain a representative sample of images containing faces, gathering a diverse and representative set of "non-face" samples is significantly more challenging. Furthermore, the small size of our available dataset compounds these difficulties, which could potentially limit the performance of even our more advanced algorithms, particularly random forest. However, we anticipate that strategies like data augmentation and careful handling of class imbalance will help mitigate these issues.

We will carefully evaluate each classifier for its strengths and weaknesses in the coming sections. Simpler algorithms like Nearest Neighbours and Logistic Regression provide useful baselines but may struggle with non-linear decision boundaries. More advanced models like SVM and Random Forest offer greater capacity for handling complex patterns, but their performance depends heavily on careful tuning and the quality of the dataset. Due to SVM's ability to handle small datasets we anticipate this to be our best performer overall.

By the end of this paper, we aim to outline the machine learning pipeline illustrated in Figure 1. This pipeline is designed to be adaptable and scalable for future improvements.

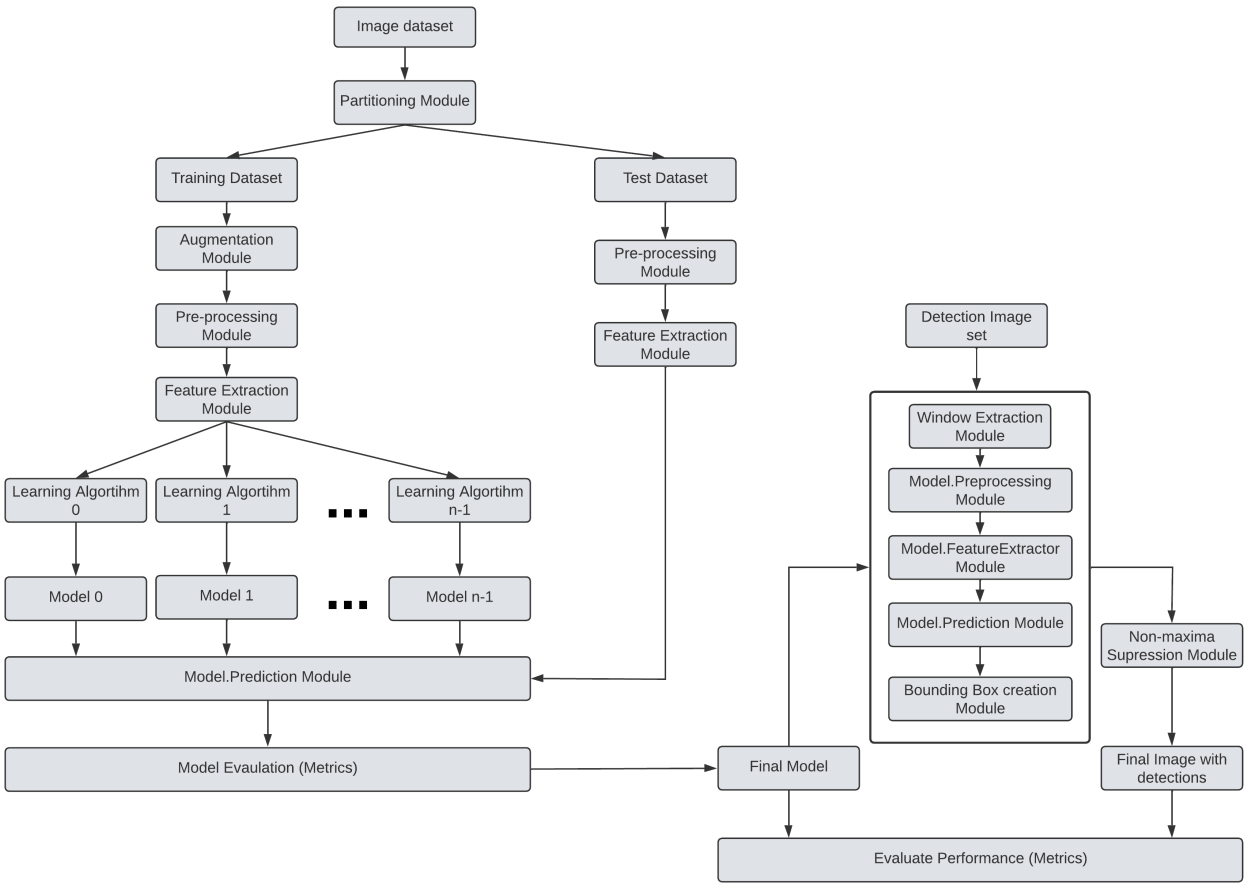


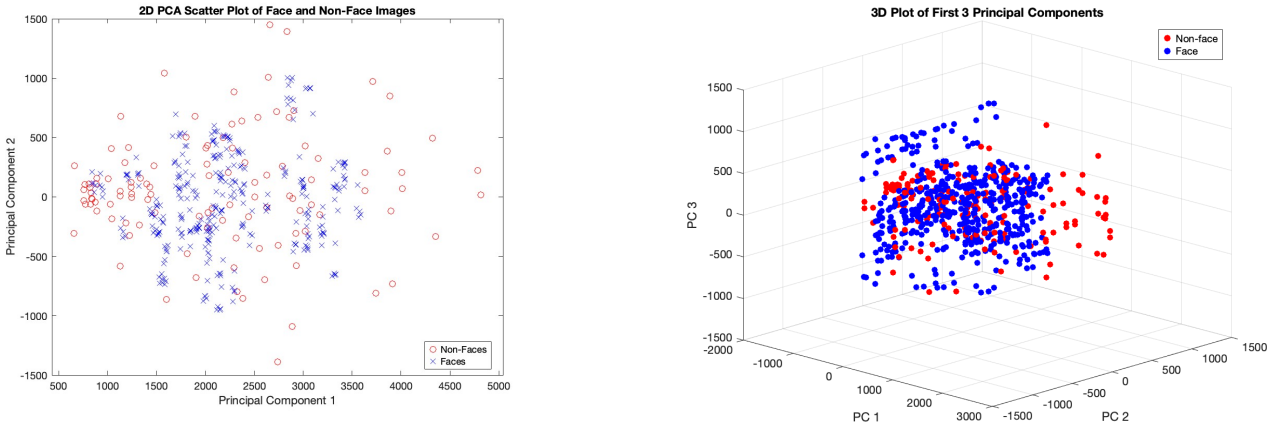
Figure 1: Proposed Machine Learning Pipeline

2 Understanding the Data

In order for us to develop an optimal model for the problem statement it is crucial we understand the data we have to train and test our model. Our data set is comprised of 124 grayscale images with spatial resolutions of 27x18, 69 of which are faces and the remaining 55 are non-faces.

Given we are performing a binary detection problem to detect faces, in an ideal world we want the face images to be completely "different" from the non-face images. That way we can easily identify a face in a real world example. So, we want to assess how different our training dataset is, however this is inherently difficult to visualise, since, we can think of our images as vectors over a 486

(27x18) dimension space. To aid in our visualisation process, we will reduce the dimensionality of the feature vector from 486D to 2D and 3D spaces. Principal Component Analysis (PCA), a widely used dimensionality reduction technique, is employed to identify and project the data onto new axes (principal components) that maximise variance in the dataset. These principal components represent the directions of greatest variability, allowing us to effectively summarise the data while retaining as much information as possible. [1]



(a) Dimension Reduction to 2D Space

(b) Dimension Reduction to 3D Space

Figure 2: PCA 2D & 3D

Some observations from Figure 2: the wide spread along the x-axis and scattered distribution, combined with the lack of distinct clusters in the first two principal components, suggest that while a significant amount of variance in the data is captured, it may not effectively separate face from non-face features. Introducing the third principal component shows some improvement in distinguishing the two classes in certain cases, but there is still a strong correlation between feature values for the two classes.

Looking ahead, simplistic decision boundary models such as Nearest Neighbours and Logistic Regression are likely to struggle in segmenting the two classes when using raw pixel values or their dimensionally reduced forms. To overcome this, we may need to incorporate an additional feature extraction step designed to enhance class separability rather than merely optimising variance. Afterward, PCA can be applied to further reduce dimensionality and eliminate redundant information from the extracted features. Advanced models like SVM, which can handle non-linear decision boundaries, should perform better but will require careful parameter tuning and the use of an appropriate kernel.

When using PCA in practice, we will extend the dimensionality until reaching a satisfactory level of cumulative explained variance, such as 0.95. For example, as shown in Figure 3, raw pixel images achieve this with 14 principal components.

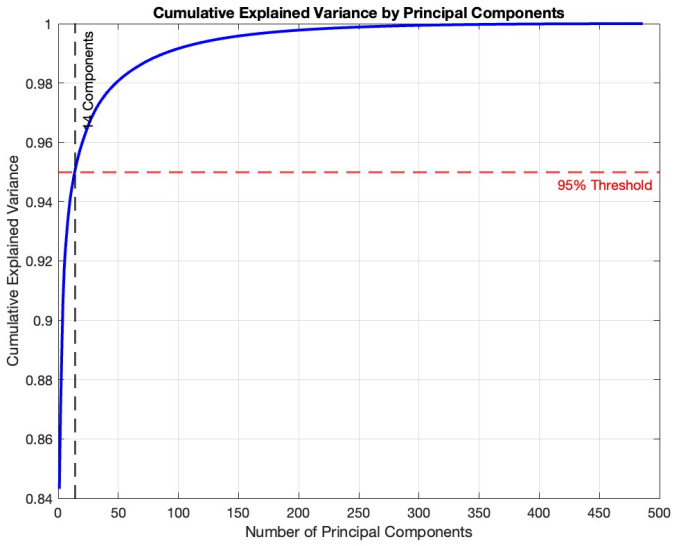


Figure 3: Cumulative Explained Variance for different PC's

Figure 3 indicates that using 14 principal components preserves 95% of the variance, reducing the feature vector size from 486 to 14 while retaining most relevant features for training our models. This demonstrates the high level of redundancy in raw pixel data.

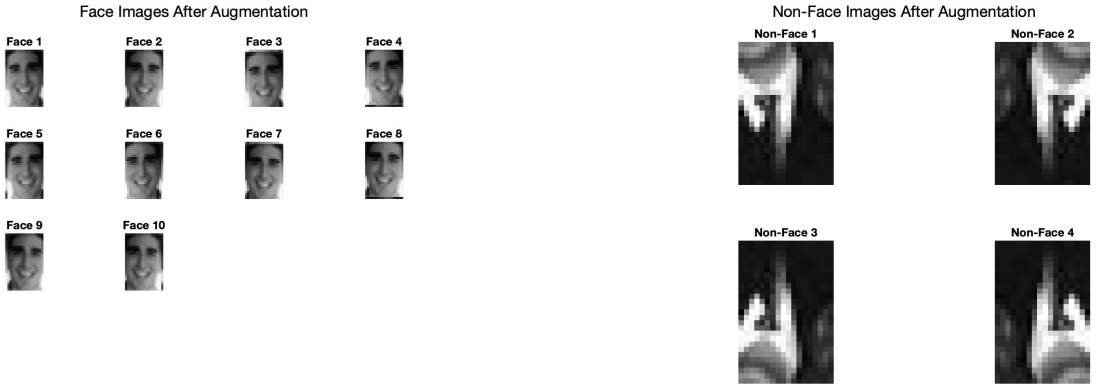
3 Data Preprocessing

Data preprocessing is an essential step in developing a robust machine learning algorithm. It serves as a prerequisite to feature extraction, streamlining the process and enhancing the accuracy and effectiveness of identifying meaningful features. By facilitating the recognition of relationships between variables, data preprocessing plays a critical role in improving the predictive performance of machine learning models.

3.1 Artificial Data Augmentation

To enhance the size of our dataset, we applied artificial data augmentation techniques to the images. This involved generating variations of the original images through methods such as mirroring, flipping, and spatial displacements. These augmentation strategies aim to improve the generalisation

capabilities of our models by increasing the diversity of the training data, thereby reducing the risks of overfitting and underfitting. The expanded dataset provides a more robust foundation for training, enabling the model to better adapt to unseen data during evaluation and deployment.



(a) Face After Augmentation

(b) Non-Face After Augmentation

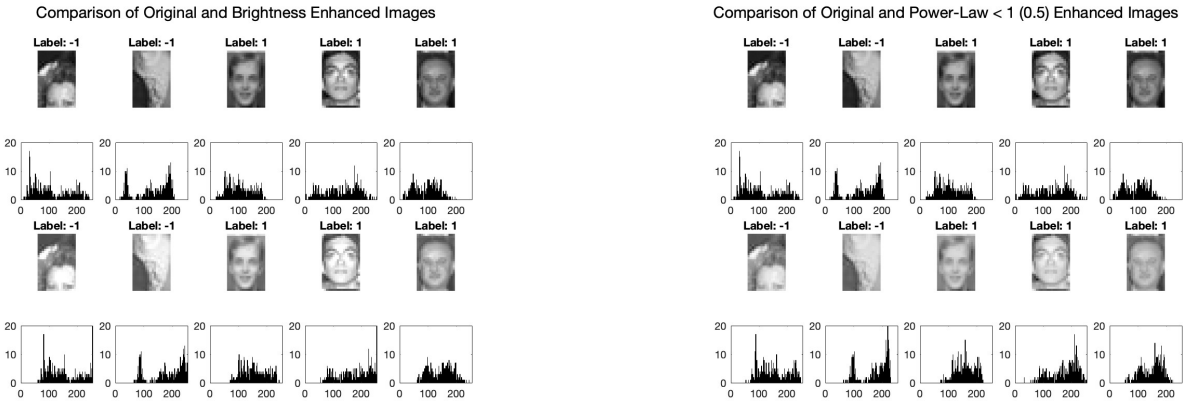
Figure 4: Face and Non-Face after Augmentation

As illustrated in Figure 4, the facial images have been augmented tenfold using techniques such as slight spatial displacements and mirroring. In contrast, the non-facial images have been augmented fourfold, employing methods like mirroring and flipping. These augmentation strategies ensure a balanced and enriched dataset, enhancing the model's ability to generalise effectively across diverse scenarios.

3.2 Image Pre-processing Techniques

Figure 5 below illustrates the original raw images (top row) alongside their corresponding histograms (below), followed by the enhanced images (middle row) with their respective histograms (below). These are shown for two data preprocessing techniques: Brightness Enhancement, Figure 5(a), and Power Law Transformation with $\gamma < 1$ in Figure 5(b). These techniques are grouped together as they are unsuitable for the requirements of this analysis.

Brightness enhancement, applied linearly, does not improve image contrast, limiting its utility in our use case. Similarly, power law transformations are effective only for images that are predominantly dark or bright, reducing its general applicability. Moreover, neither technique is adaptive; they lack automatic configuration to suit each image within a dataset. Consequently, introducing a new image to the dataset risks unpredictable effects, potentially hindering rather than supporting the feature



(a) Brightness Enhancement

(b) Power Law < 1

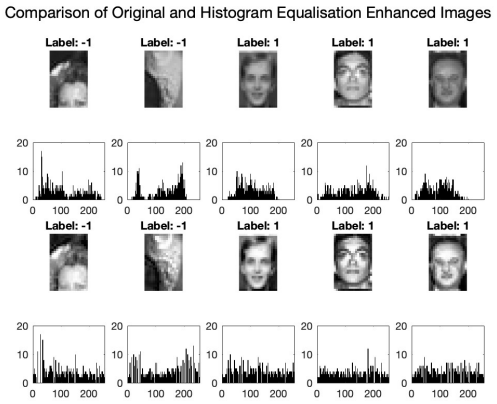
Figure 5: Brightness Enhancement & Power Law < 1

extraction process.

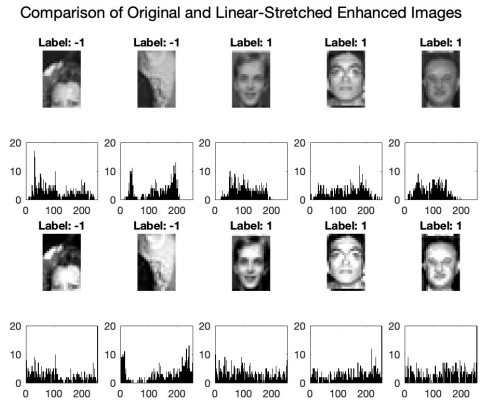
Figure 6 below showcases four preprocessing techniques: Histogram Equalisation (Figure 6(a)), Linear Stretching (Figure 6(b)), Mean Filtering (Figure 6(c)) and Median Filtering (Figure 6(d)).

Both Histogram Equalisation and Linear Stretching have been identified as suitable techniques for our machine learning pipeline due to their ability to automatically adjust image contrast across a wide range of images. This adaptability ensures that when new data is introduced, these techniques can effectively process the data to optimise feature extraction. By enhancing contrast these techniques redistribute pixel intensities across the available range, making image features more distinct and detectable, thereby improving model performance. Among these, Histogram Equalisation was selected as the preferred preprocessing technique as it yielded the best results for feature extraction.

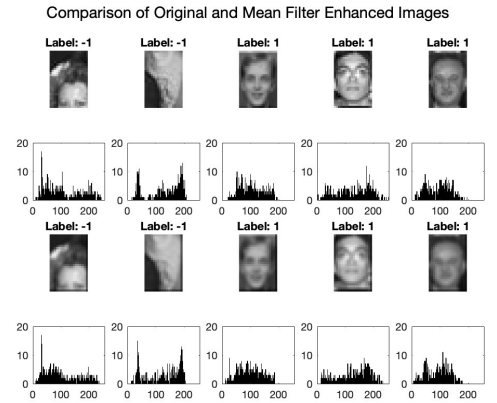
The filtering techniques are included to enhance feature extraction in our machine learning pipeline. The mean filter, while effective in noise reduction, tends to smooth out edges, resulting in a loss of finer details. In contrast, the median filter retains more critical details while effectively removing high-frequency noise. Given its superior ability to preserve image detail while reducing noise, the median filter is better aligned with the needs of our feature extraction process.



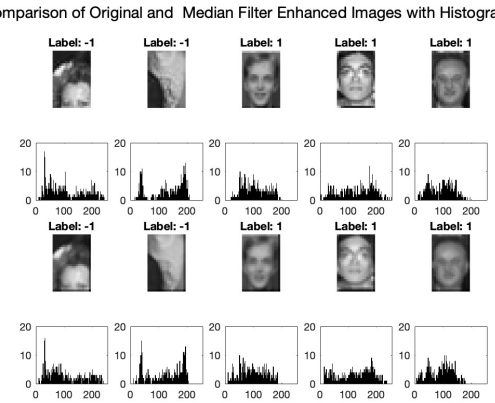
(a) Histogram Equalisation



(b) Linear Stretching



(c) Mean Filter



(d) Median Filter

Figure 6: Histogram Equalisation, Linear Stretching Mean Filter And Median

4 Feature Descriptors

Feature descriptors are fundamental in image pre-processing for machine learning pipelines, offering a compact and meaningful representation of an image's content. By extracting and encoding essential patterns, such as edges, textures, or key points, descriptors simplify complex image data, making it more manageable for computational analysis while preserving the most relevant information. In this work, we experiment with seven feature descriptors to identify an effective approach for capturing the critical elements of an image in a low-dimensional format, facilitating efficient and accurate processing.

4.1 Raw pixel values

The most basic and intuitive approach of representing an image as a feature vector is to simply pass in each of the images in the form of a flattened vector of dimension imageWidth x imageHeight. Every entry represents the intensity value of each pixel in an image. In our case this means passing a vector of dimension 486 containing gray scale values with a dynamic range of 256 (i.e values between 0-255). Below in Figure 7 is a subset of images in their raw pixel form.

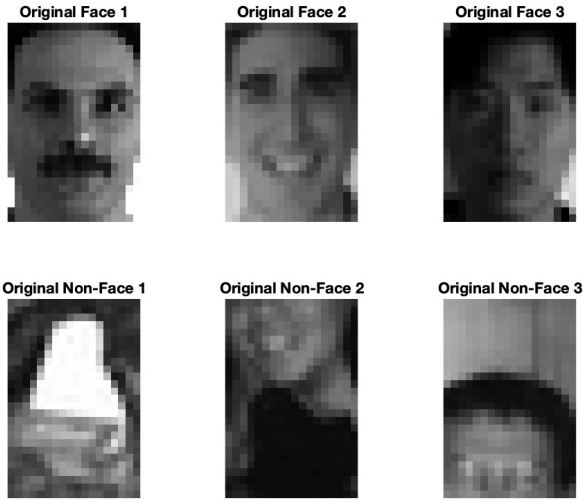


Figure 7: Raw-Pixel Image Representation

While to the human eye this appears to be the logical approach, machines work better when we give them strictly relevant information. In general it is best practice to reduce the dimensionality of the raw-pixel feature vector before training the machine. This help with computational cost and to avoid over-fitting to noise.

4.2 PCA - Dimensionality Reduction

We previously discussed the benefits of dimensionality reduction in Section 2; however, we now take a closer look at what each of the principal components represents within our dataset. In Figure 8, we illustrate the first 16 eigenvectors trained on the raw pixel inputs of our image training set.

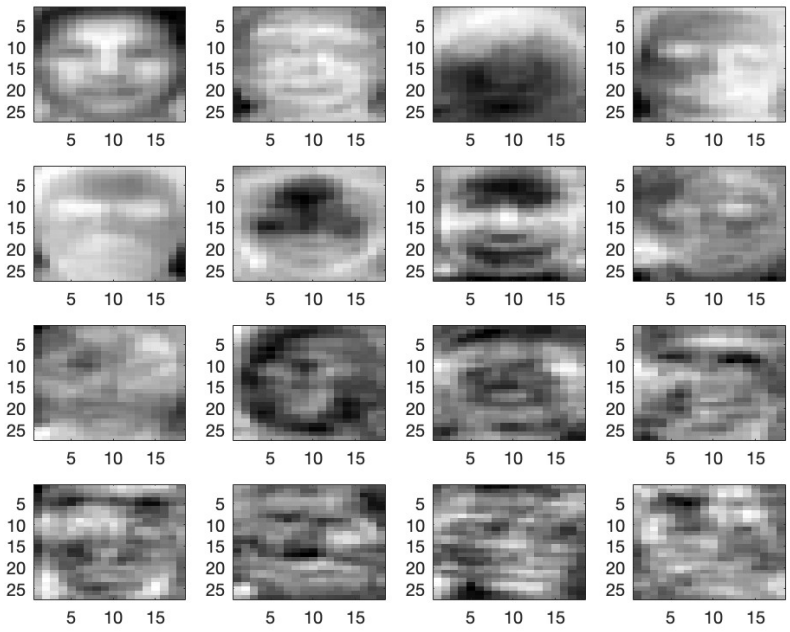


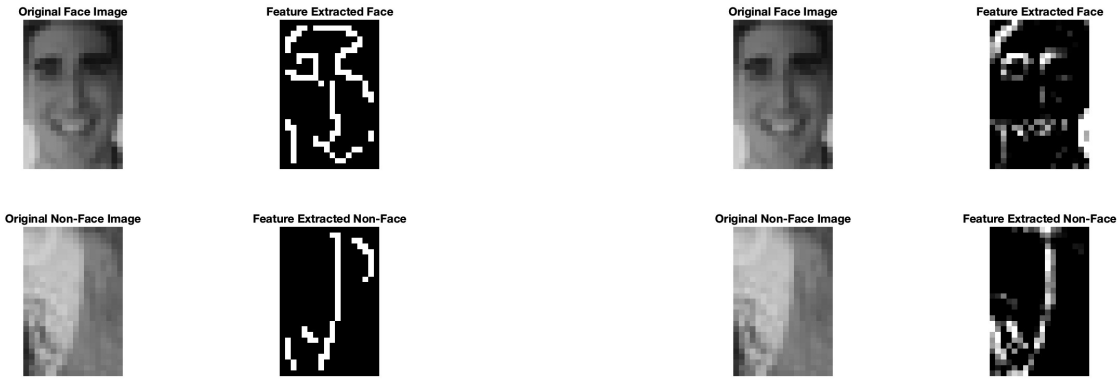
Figure 8: First 16 eigen-vectors of PCA

Our earlier claim that 14 principal components capture 95% of the variance becomes clearer in this context. From Figure 8, we can see that even with just the first principal component, we effectively capture the essential structure of a face. As we progress to later components, the features they capture become increasingly abstract and less visually interpretable. Referring back to Figure 3 in Section 2, we observed that the first component alone accounts for approximately 84% of the variance, reinforcing this observation. Together, these points highlight the advantages of reducing the dimensionality of our feature vectors, enabling us to retain meaningful information while simplifying the dataset before training our model.

4.3 Edges

In facial recognition, edge detection helps extract edge maps from facial images, aiding in noise reduction and the identification of fine details. Initially, we chose one of the most effective methods: Canny Edge Detection. However, we had some doubts for our given task, which will be discussed later. To explore alternatives, we also compared it with a simpler method - calculating the gradients in the x and y directions and determining the gradient magnitude. The results of both approaches are

presented in Figure 9.



(a) Canny Edge Extraction (Binary) (b) Edges Gradient Extraction (Continuous)

Figure 9: Canny Binary & Edge Gradient Continuous

As illustrated in Figure 9(a), when using the edge extraction function provided by MATLAB for Canny edge detection, the output is always a binary image. While this is effective in detecting the presence of edges, it overlooks the intensity of the edges. The Canny algorithm typically highlights edges based on both their presence and strength, providing a richer, more informative result. In contrast, a binary image treats all detected edges equally, regardless of their intensity, which may not always be ideal.

With this in mind, we experimented with our own alternative edge extraction method based on image gradients, where the magnitude of these gradients is calculated. This approach, shown in Figure 9(b), provides us with a continuous numerical value to represent edge strength, with more intense edges being emphasised relative to weaker ones. This allows us to capture more nuanced edge information, rather than just detecting the presence of an edge. However, this method is not as robust as the Canny edge detector, which includes additional steps to improve its accuracy and reliability. Canny Algorithm:

1. Noise Reduction through Gaussian filtering
2. Gradient Calculation
3. Non-maximum Suppression to thin edges
4. Edge Tracking by Hysteresis to attempt to suppress weak edges

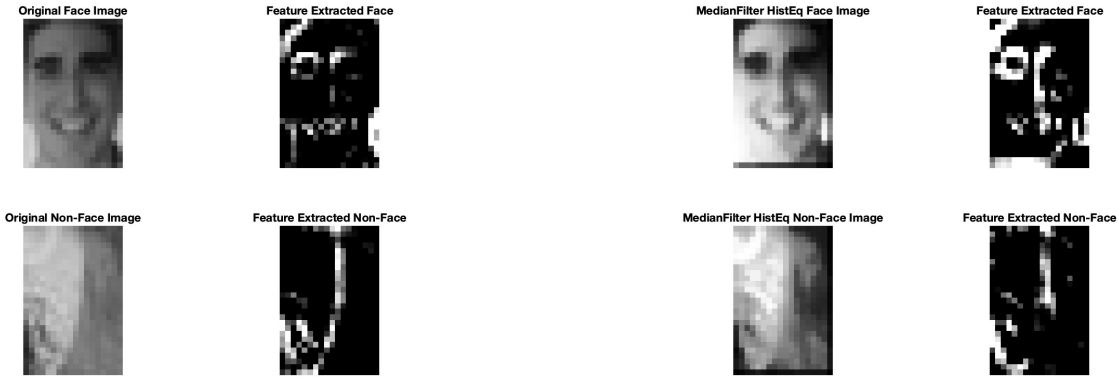
By default the edges filter with 'Canny' will keep the threshold value to 1, so our step 4 is not very effective here. While we could improve our method by experimenting with different thresholds, this

would require additional experimentation to identify the most suitable parameters for our data which may be hard to definitively capture for unseen data.

To address some of the gaps in our approach, we introduced several enhancements to make our edge extraction process more robust. Specifically:

1. Median filter to reduce noise.
2. Histogram equalisation to compensate for illumination variations.
3. Gradient magnitudes to give a continuous values and highlight stronger edges.
4. PCA to reduce dimensionality and retain the most relevant information.

The results of which are shown below in Figure 10.



(a) Edges Gradient Extraction (No Preprocessing) (b) Edges Gradient Extraction (Preprocessing)

Figure 10: Edge Extraction after Preprocessing

We can see from Figure 10(b) that we more effectively capture key elements of a face such as the nose and mouth in comparison to 10(a). We will experiment with both our Canny edges and our proposed edge extraction algorithm and gain results for both in the future sections.

4.4 LBP - Local Binary Patterns

In the initial phase of our research Local Binary Patterns (LBP) emerged as an attractive feature extraction method as it is a powerful texture descriptor commonly used in facial detection and recognition systems. It transforms facial images into a compact feature representation by analysing local texture patterns. Texture features are fundamental to image analysis because they capture repeating patterns, local variations, and spatial arrangements of intensities that characterise different surfaces and objects.

In facial detection applications, LBP excels at capturing local texture patterns by encoding the relationships between pixel intensities in small neighbourhoods. This process creates distinctive binary descriptors that effectively represent skin texture, facial micro-patterns, and surface variations, which maintain consistency even under varying lighting conditions. These binary patterns are subsequently converted into histograms, providing an efficient mechanism for facial feature classification.

Our implementation of LBP faced challenges due to the limited resolution of our input images (27x18 pixels). This constraint necessitated careful consideration of the cell size for our LBP implementation.

Overall we believe LBP could have performed better given an improved dataset that included higher resolution images.

Cell Size Analysis

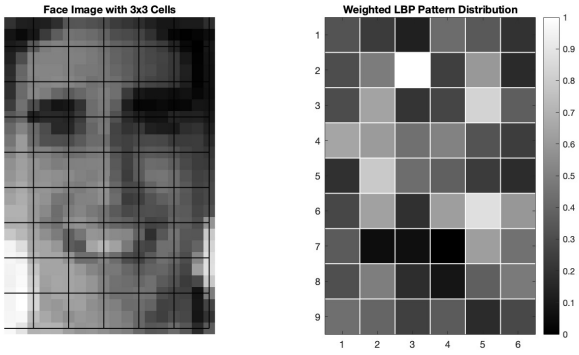


Figure 11: LBP implementation [3,3] cell array

A 3x3 cell configuration in LBP offers several strengths, provides higher spatial resolution to detect facial features, allows better preservation of local feature positions, allows for more precise localisation of key facial regions such as eyes, nose, and mouth, and enables finer granularity in texture pattern detection. However, this small cell size also presents limitations: each cell contains only 9 pixels for

analysis, making it more susceptible to noise, the feature representation is less stable due to limited pixel information, larger-scale texture patterns may be missed, and the analysis becomes more sensitive to minor variations in pixel intensity. This fundamental trade-off between detailed feature capture and noise sensitivity makes the 3x3 cell size most suitable for applications requiring fine-grained facial feature detection, where image quality is high and noise is minimal.

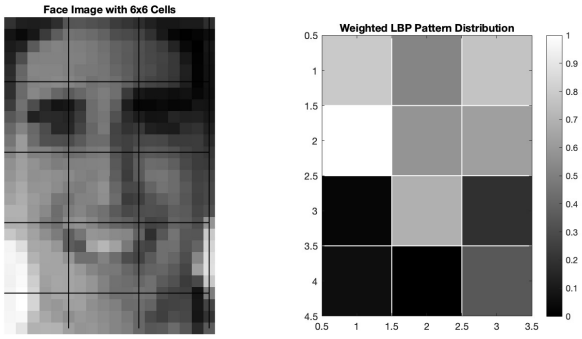


Figure 12: LBP implementation [6,6] cell array

A 6x6 cell configuration in LBP facial analysis produces a 4x3 grid of cells, offering better pattern detection stability with 36 pixels per cell compared to smaller configurations. This larger cell size provides better noise resistance and greater texture representation due to its increased sampling area. However, the reduced spatial resolution of only 12 regions results in less precise feature localisation, as multiple facial features may merge within the same cell. This makes it more difficult to accurately distinguish between key facial regions like eyes, nose, and mouth, potentially leading to feature overlap and loss of fine detail detection. The 6x6 configuration is thus most suitable for applications prioritising stability and noise resistance over precise facial feature localisation.

The optimal balance between cell configurations and their impact on facial recognition performance will be investigated in our parameter tuning section further.

Comparative Analysis

When compared to high-resolution implementations like [2], who utilised 7x7 grids on higher resolution images, our results demonstrate the significant impact of resolution on LBP effectiveness.

Figure 2: The original face image *(left)* followed by the weighting scheme for the *7×7* cells *(right)*.

Figure 13: [7x7] cell array implementation on high resolution image

Results

Despite these limitations, our LBP implementation successfully captured some facial structures, particularly around the eye regions. However, the feature extraction quality would be substantially improved with higher resolution input images.

4.5 HOG - Histogram of Gradients

In facial detection applications, HOG excels at capturing structural and edge information by computing gradient magnitudes and orientations of the image pixels. This process creates a dense grid of histograms that effectively represent facial contours, edges, and structural variations, which maintain consistency under varying lighting conditions, moderate viewpoint changes, and local geometric transformations. This is achieved through the gradient computation and block normalisation process, which makes HOG a good choice for our feature extraction pipeline.

Cell and Block Structure

In our facial recognition implementation, we process 27x18 pixel images using an 8x8 pixel cell size organised into 16x16 pixel blocks, with each block containing 2x2 cells. The blocks use an 8-pixel stride, creating 50% overlap between adjacent blocks, ensuring feature capture and smooth transitions across the image regions.

The visualisation below shows how we utilised HOGS fundamental principals: Edge detection through gradients, Local feature capture in cells, Normalisation across blocks and Orientation binning through

gradient directions

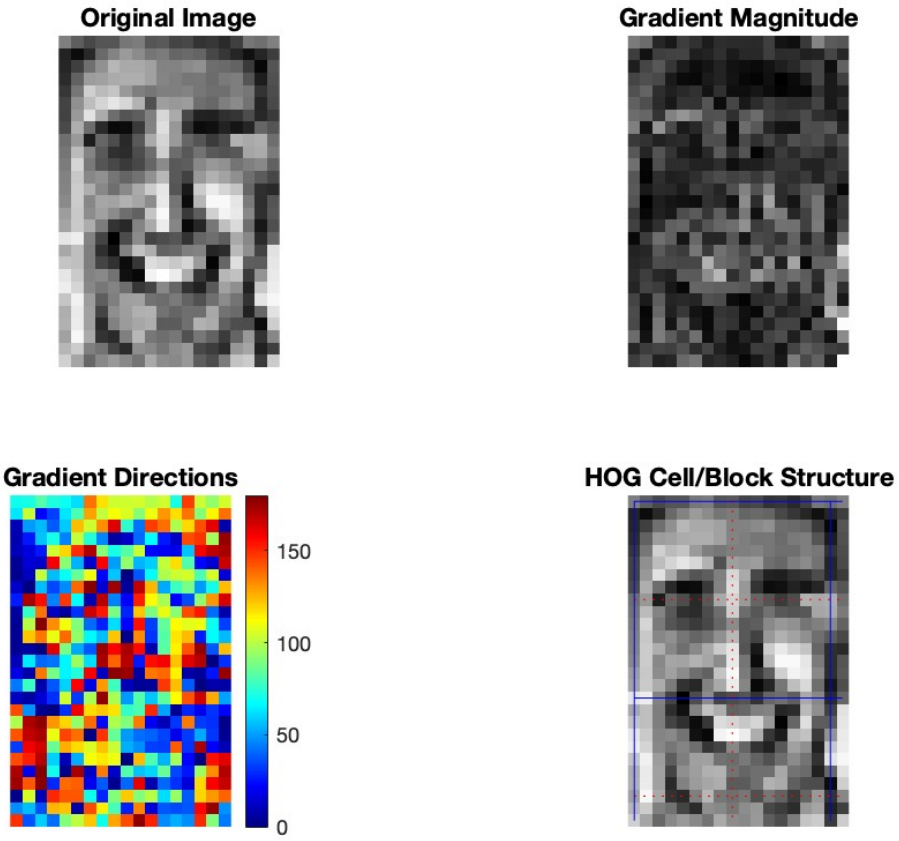


Figure 14: Visualisation of HOG feature extraction components showing: (a) Original image (b) Gradient magnitude (c) Gradient directions (d) Cell/Block structure.

HOG Feature Visualisation Analysis

Original Image (a)

The original image preprocessed using histogram equalisation shows the face image at 28x17 pixels resolution, Its a grayscale representation where brighter pixels represent higher intensity values. There are clearly visible facial features including eyes, nose, and mouth despite the low resolution

Gradient Magnitude (b)

The gradient magnitude shows the strength of edges in the image, the brighter areas indicate stronger gradient changes and the dark areas indicate minimal changes in the pixel intensity. Clearly highlighting facial features including Strong edges around eyes, nose and mouth contours

Gradient Directions (c)

Here is a colour-coded representation of gradient orientations from 0 to 180 degrees showing how the edges are oriented across the face with the different colours representing different edge directions.

HOG Cell/Block Structure (d)

The Cell/Block structure shows how the image is divided into cells and blocks with the red dotted lines representing 8x8 pixel cells, the blue solid lines representing 16x16 blocks (2x2 cells). This demonstrates the hierarchical structure with each cell capturing gradient information, blocks group cells together for normalisation and blocks overlap for better feature representation.

4.6 Gabor

In facial detection applications, Gabor filters excel at capturing a wide range of details by effectively analysing both high and low-frequency components of an image. At lower scales, they focus on high-frequency features such as sharp edges and fine textures, while at higher scales, they capture low-frequency elements like broader edges and larger structural patterns. Simultaneously, the filters adapt to various orientations, from horizontal to vertical and diagonal, enabling the detection of gradients and edges from multiple perspectives. This multi-scale and multi-orientation capability makes Gabor filters particularly well-suited for extracting the diverse and intricate features required in facial detection.

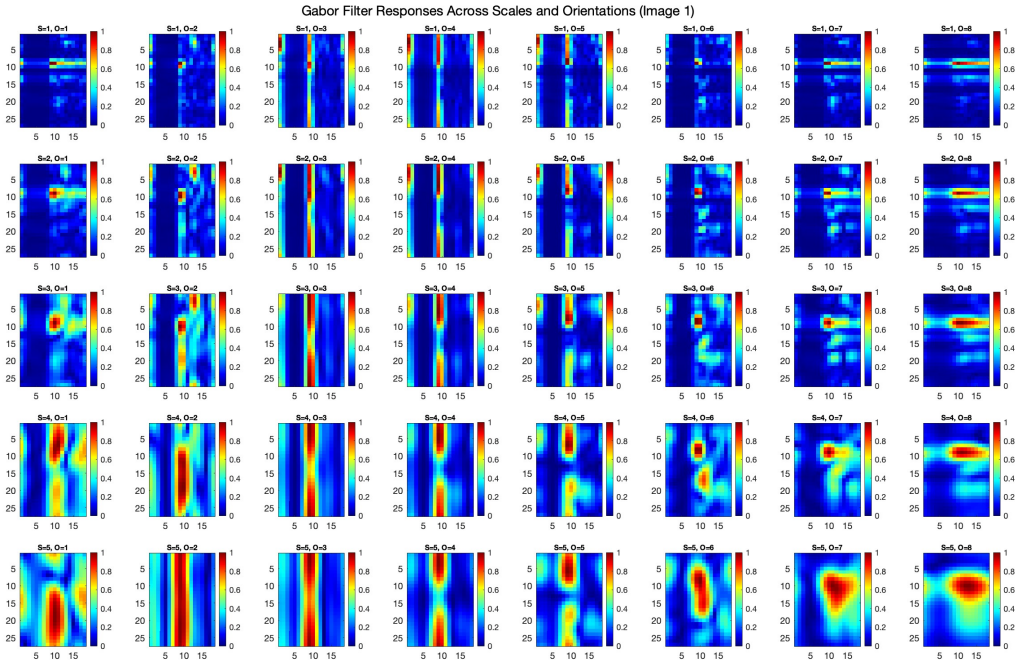


Figure 15: Gabor Filter S=1:5 & O=1:8

The figure above illustrates the Gabor filter responses for the first facial image, applied across multiple scales (S=1 to S=5) and orientations (O=1 to O=8). At lower scales (S=1 and S=2), the Gabor filters capture high-frequency components in the image. These responses are characterised by bright regions that are highly localised and concentrated in smaller areas, reflecting the filter's sensitivity to finer details. Conversely, at higher scales (S=3 to S=5), the filters capture low-frequency components. The responses at these scales are more diffused, with bright regions spreading over wider areas, demonstrating the filters' focus on capturing more global features.

Regarding orientation, the filter responses are sensitive to specific edge directions or gradients. For example, filters at O=1 are tuned to detect horizontal patterns, O=5 focus on vertical structures, and O=3 and O=7 highlight diagonal gradients. Bright regions in these subplots indicate areas of the image that strongly align with the respective orientation of the filter. This directional sensitivity enables the Gabor filters to effectively highlight patterns and gradients in various orientations, capturing critical edge information from multiple perspectives. This Enables edge detection, texture analysis, and facial feature extraction.

We expect Gabor features to perform poorly due to their extremely high dimensionality, with 19,440

features in this case, which can completely overwhelm the model. Many machine learning models struggle to generalise effectively in such high-dimensional spaces, especially when dealing with features that are redundant or noisy. Furthermore, Gabor features are highly sensitive to image variability, such as changes in lighting, pose, and expression. These variations can significantly alter the extracted Gabor features, reducing their consistency and making it difficult for models to identify meaningful, highly correlated patterns.

To make the feature set more manageable and feasible for training, PCA was applied as a dimensionality reduction step. However, PCA did not improve the performance of Gabor features because it focuses on maximising variance rather than explicitly removing irrelevant or noisy components. Gabor features tend to be highly redundant and non-linear, which PCA is not well-equipped to handle effectively. As a result, PCA often retains irrelevant high-variance components while discarding subtle but important features critical for classification, further limiting the model's ability to generalise effectively.

5 Training and Parameter Tuning

Before testing our model on the detection we need to optimise our choices of parameters to ensure we are giving the model the best chance at performing well. We developed a systematic automated pipeline to perform hyper-parameter tuning for us. To achieve this we deployed Grid Search Hyper-parameter Optimisation in combination with a 5-fold stratified cross validation and Test-Time Augmentation pipeline illustrated in Figure 15. To understand the process more clearly, it is defined as follows:

1. Define a set of hyper-parameters we want to test our model with.
2. Repeat the following for each hyper-parameter combination:
    - Split training set using K-Fold stratified cross validation.
    - Train a model on the training set for each fold.
    - Use Test-Time Augmentation on the Test Set and take the majority class vote for each image.
    - Evaluate performance of model with current hyper-parameters.
3. Compare each of the models performances and take the best performing model and the hyper-parameters associated with it.

4. Re-train the model on the entire augmented training set with the best hyper-parameters.

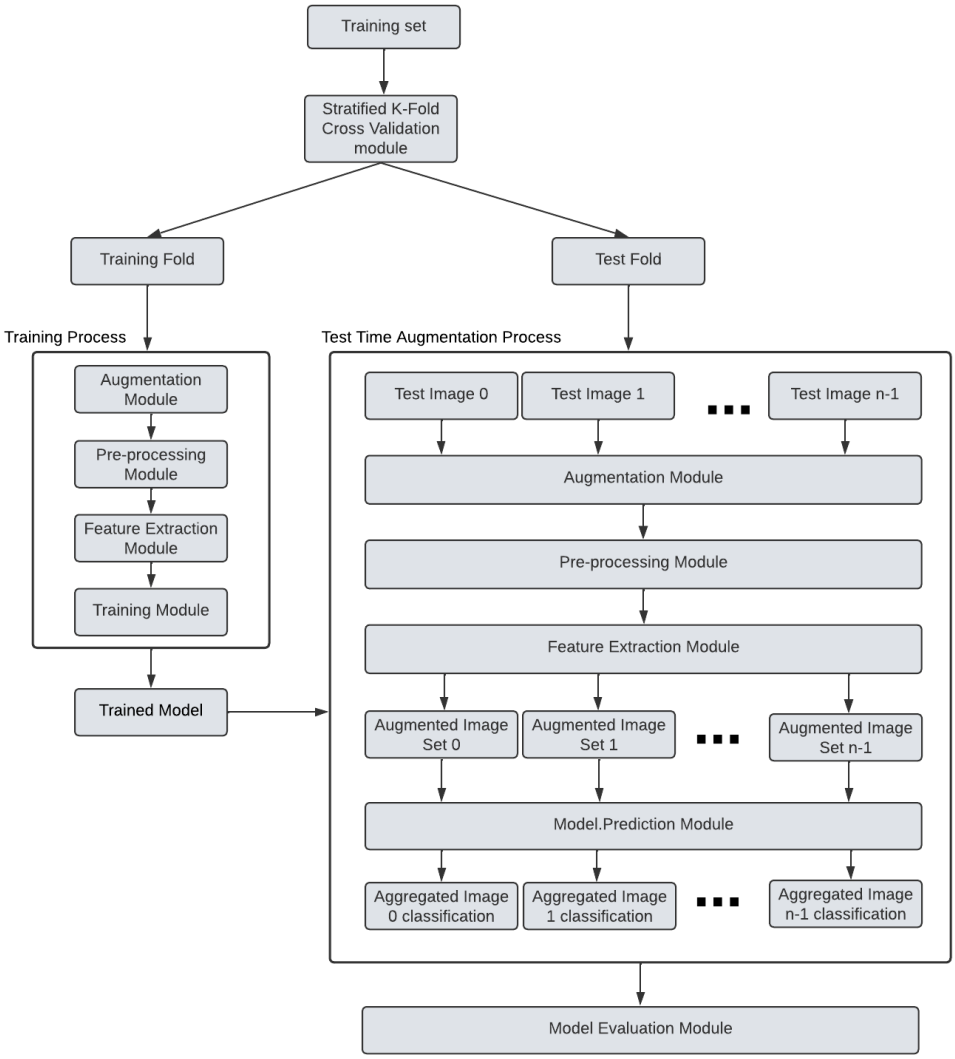


Figure 16: Cross Validation & TTA Pipeline

We will explain in some more depth what each of the key components of this pipeline are doing.

5.1 Stratified K-Fold Cross Validation

In K-fold cross-validation, the data is divided into K folds; the model is trained on K-1 folds and tested on the remaining fold. This process is repeated K times, and the results are averaged to give a more robust estimate of the model's performance. This is shown below in Figure 17.

We opted for cross validation to partition our data set as, in general, it is preferred over a simple train-test split since it reduces the risk of overfitting and provides a more reliable measure of generalisation. A single train-test split might lead to biased results if the split isn't representative of the entire dataset. Cross-validation, by training and testing on different subsets, ensures that the model is evaluated on all data points, improving the stability and reliability of performance estimates.

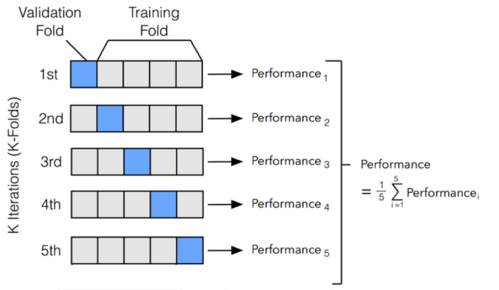


Figure 17: K-Fold Cross Validation

To make this even more robust we used stratified sampling on 5-fold cross-validation which is most suitable for an image set of 69 faces and 55 non-faces because it ensures that each fold maintains the same class proportion as the original dataset. This is important given the relatively small dataset and slight class imbalance. By preserving the ratio of faces to non-faces in each fold, stratification reduces the risk of performance bias and provides a more reliable estimate of the model's ability to generalise across both classes, especially in tasks involving imbalanced datasets like face detection.

5.2 Test Time Augmentation

Data augmentation as we have previously described is the expansion of a dataset by adding transformed copies of each example - it is a common practice in machine learning. Typically, data augmentation is performed when a model is being trained. However, it can also be used at test-time to obtain greater robustness, improved accuracy, or estimates of uncertainty. Test-Time Augmentation

(TTA) entails pooling predictions from several transformed versions of a given test input to obtain a "smoothed" prediction. [3] In our case we applied it in the following way:

1. Augment a given test image.
2. Test the model on the augmented set of the test image.
3. Take the majority vote of the class and use that as the predicted label of the original test image.

Our primary goal with TTA was to ensure that final predictions were robust to any single unfavourable augmentation, thereby avoiding biased testing results. For instance, consider a scenario where we test a model on a subset of 5 faces and 5 non-faces. Augmenting the test set would result in 50 faces and 20 non-faces. In such a case, if the model simply predicts "face" every time, it would achieve an accuracy of 71.4%, which is misleading. However, by applying TTA and using majority voting across augmentations, the accuracy would more accurately reflect performance at 50%, offering a fairer evaluation. Alternatively, avoiding augmentation entirely may overlook situations where a model predicts the wrong class with low confidence (close to 0.5). TTA addresses this by aggregating votes from multiple augmentations, increasing the likelihood of correct classification through consensus.

5.3 Grid Search Hyper-parameter Optimisation

One of the most widely used methods to learn hyperparameter configuration space is grid search (GS). Grid search can be described as an exhaustive exploration or a method of brute force that tests all the combinations of hyper parameters given to the grid configuration. GS alone will not further exploit the well-performing areas. Therefore, the following process needs to be done manually to identify the global optimums:

1. Start with a wide space for searching and phase scale.
2. Based on the previous findings of well-performing hyperparameter environments narrow the search space.
3. Repeat step 2 several times before the optimum has been attained.

However, the major down-side of GS is its ineffectiveness in the configuration space of high-dimensionality hyperparameters, as the number of assessments increases exponentially as the hyperparameter frequency increases. This exponential progress is known as the dimensionality curse. Assuming that k

parameters exist and each of them has n separate values, its computational complexity increases exponentially at a rate of O(n^k). Thus, GS can be an efficient HPO approach only when the hyperparameter configuration space is limited. With this in mind, we took a pretty basic approach to GS, for example for Random Forests we only optimised the number of trees and didn't bring other hyper-parameters such as number of predictors or class weight into the picture. [4]

Due to the time complexity of the grid search algorithm we first narrowed down which feature combinations were worth bringing forward for each learning algorithm to save our resources. To do this we chose a reasonable parameter choice (e.g K = $\sqrt{\text{num\_Images}}$ for KNN) and ran the pipeline in Figure 15 for each feature type. The relative best performers were then further parameter tuned using Grid Search Hyper-parameter Optimisation as described above. Finally, we then take the best performing model(s) and trained them on the entire image set before then bringing them into the detection task.

6 Model Selection

One of the most important, and often difficult, decisions to make when creating a machine learning pipeline is what model to use for the problem. The first step of this project will be to sensibly decide on a final model based on model selection techniques such as probabilistic measures and resampling methods [5]. To begin with we will use Nearest Neighbour, K-Nearest Neighbour (KNN), Support Vector Machine (SVM), Logistic Regression and Random Forest as our initial models and compare their performance accordingly. These supervised machine learning algorithms are common choices for binary classification tasks, with some of them expecting to perform better than others as mentioned in our opening chapters.

6.1 Random Forest

Bagging and Classification

Random Forest is a versatile machine learning model widely used for both classification and regression tasks. It makes predictions by constructing an ensemble of decision trees, where the final output is determined by a majority vote for classification or by averaging the predictions for regression. In our approach, we opted for classification and bagged trees, which utilise subsets of features rather than the entire feature set. This technique reduces variance, minimises overfitting, and filters out noise, ultimately improving the model's ability to generalise to unseen data.

Underperforming Feature Extraction Techniques

Table 1: Results Table For Random Forest NTrees = 350

	RawPix	PCA	Edges	EdgesPCA	GaborPCA	HOG	LBP
Accuracy	90.00%	93.33%	66.67%	76.67%	66.67%	90.00%	86.67%
Precision	0.90	0.91	0.67	0.74	0.67	0.87	0.83
Recall	0.95	1.00	1.0	1.0	1.00	1.00	1.0
F1 Score	0.93	0.95	0.80	0.85	0.8	0.93	0.91
TP	19	20	20	20	20	20	20
TN	8	8	0	3	0	7	6
FP	2	2	10	7	10	3	4
FN	1	0	0	0	0	0	0

Based on the data in Table 1, some feature extractors perform exceptionally well, achieving accuracies above 90%. However, a closer examination reveals that methods such as Edges, Edges PCA, and Gabor PCA exhibit low accuracy and high false positives (Type I errors). This means these methods often misclassify non-faces as faces, which is particularly problematic in a detection model as it would result in numerous false alarms.

The poor performance of these techniques may stem from their tendency to capture irrelevant patterns in the data. Interestingly, applying PCA to edge features significantly improves performance, suggesting that dimensionality reduction played a role. However, this improvement is unlikely due to dimensionality alone, as PCA did not enhance Gabor features. Instead, it is more plausible that PCA removed redundant information and generated a set of uncorrelated components, making it easier for the model to learn meaningful patterns.

While LBP and Raw Pixels achieved relatively good accuracy, they are less promising for further exploration. PCA provides a more refined version of raw pixel data, and HOG consistently demonstrates the best accuracy overall. As such, HOG emerges as the most effective feature extraction technique for this task.

Hyper parameter tuning best models

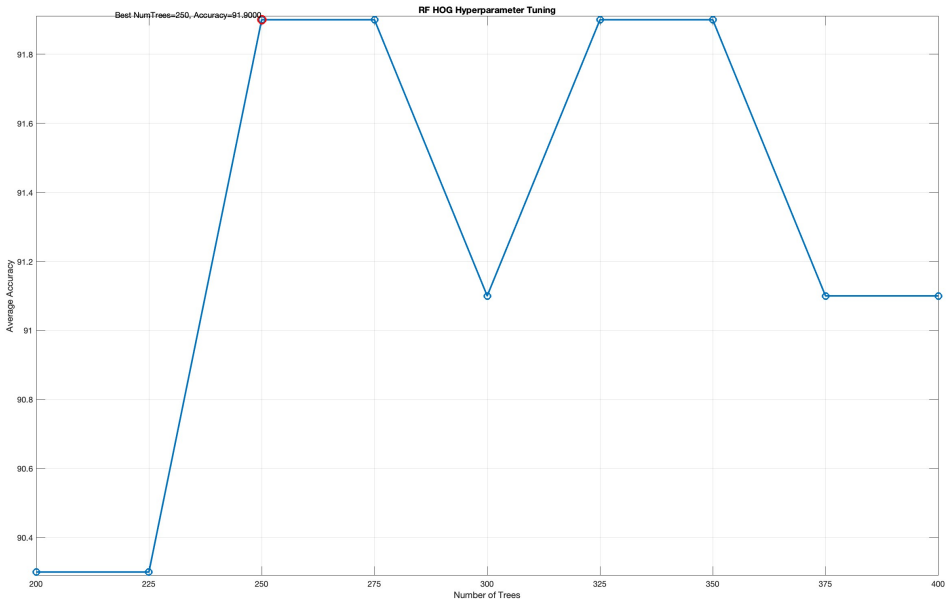


Figure 18: RF HOG Grid Search

Figure 18 above illustrates the grid search (K-fold cross-validation) results for a range of tree counts (200-400) using HOG as the feature extraction technique. The results confirm that HOG outperforms other techniques from previous tables, further solidifying it as the most effective feature extractor.

At 200 trees, the model appears to be underfitting the data. At this stage, the ensemble lacks sufficient diversity and stability, leading to higher variance and an inability to capture all meaningful patterns in the data, resulting in slightly lower performance. Between 250 and 350 trees, the model achieves the optimal balance between diversity and generalisation, yielding its best performance. This range represents the most effective number of trees for the model.

Beyond 350 trees, the accuracy begins to decline. This is likely due to overfitting, where the model starts learning overly specific patterns from the training data. As a result, its ability to generalise to unseen testing data diminishes, as it relies on patterns too closely tied to the training set.

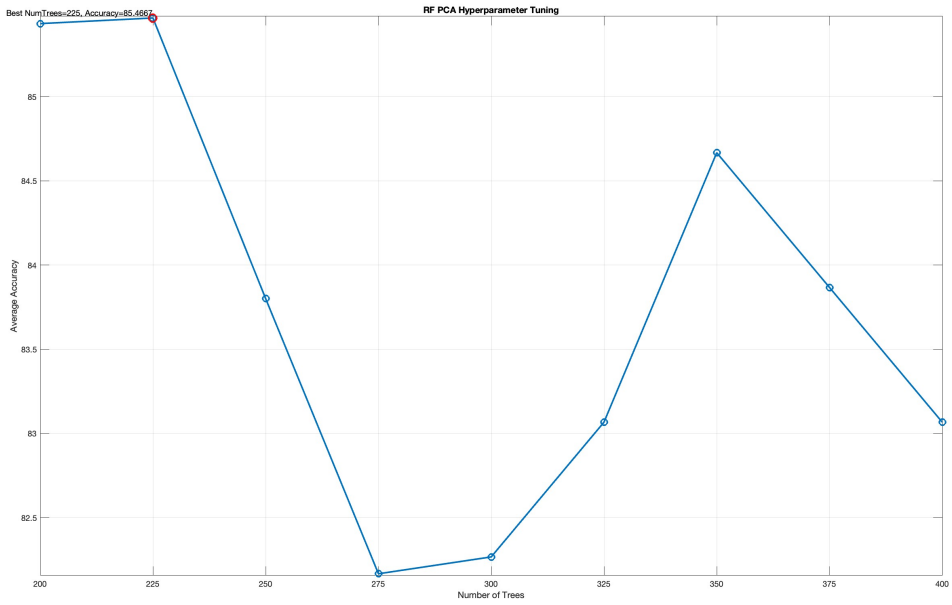


Figure 19: RF PCA Grid Search

Figure 19 above presents the grid search results (K-fold cross-validation) for a range of tree counts (200-400) using PCA as the feature extraction technique. Unlike the previous results, the accuracy has decreased, which is likely due to the use of cross-validation on the images, providing a more reliable and accurate representation of the model's true performance.

The graph shows that the accuracy peaks at 225 trees, indicating the most optimal number of trees for the model. However, the accuracy sharply declines beyond this point. This drop is likely due to PCA features containing a significant amount of redundant information and noise. Unlike other feature extraction techniques, PCA lacks the inherent structure and discriminative power needed to produce more meaningful and robust features. This may lead the model to struggle with capturing relevant patterns, resulting in reduced performance as the number of trees increases.

Performance of best Feature Extractor

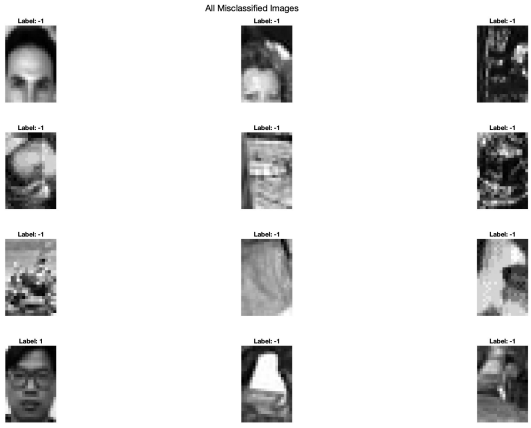


Figure 20: Random Forest Misclassified Images

When examining the performance of HOG with Random Forest, we observe from Figure 20 that the only misclassified face belonged to an individual wearing glasses. This misclassification may have occurred because the glasses disrupted the model's ability to detect typical facial features, possibly creating patterns that confused the classifier.

Regarding the non-faces incorrectly classified as faces, 2 out of the 11 were partial faces, while the remaining 9 were non-faces. The misclassification of these non-faces as faces could be attributed to several factors. One potential reason could be as simple as the stratified cross-validation process, which maintains the ratio between faces and non-faces across folds but doesn't gaurantee each training fold is the same size. Essentially meaning at particular folds we might have very little training data, particularly non-faces, and struggle to preform accurate predictions at that fold.

Alternatively, it could be the case that these non-face images simply contain a large amount of textures and patterns and some of which may correspond to similar patterns of facial features. In some of the showcased images we could attest to this, however it is not entirely clear for all of them. Such challenges highlight the importance of having a diverse and representative dataset during training to improve the model's robustness to cases like this.

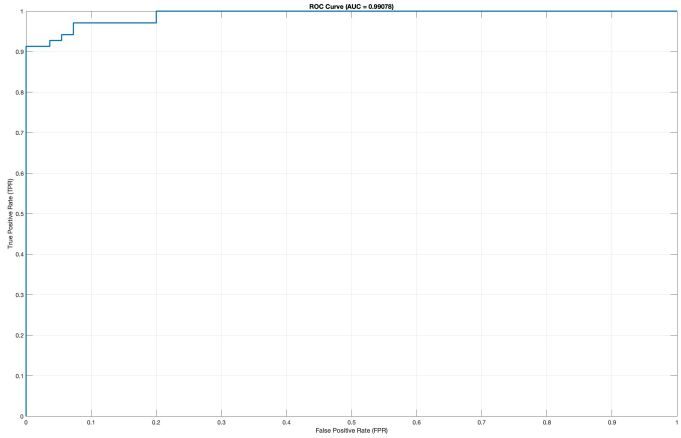


Figure 21: ROC Curve For Random Forest HOG

From Figure 21 above, the area under the curve (AUC) is 0.99078, indicating that the Random Forest model with HOG features performs exceptionally well at distinguishing between the two classes. This near-perfect AUC reflects an excellent balance between specificity and sensitivity on the test set. Given its strong performance, the model demonstrates a high potential to generalise effectively to unseen data. Therefore, we can expect it to perform well in the face detection task in the next segment.

6.2 Nearest Neighbours & K-Nearest Neighbours

Table 2: Results Table For Nearest Neighbours

	RawPix	PCA	Edges	EdgesPCA	GaborPCA	HOG	LBP
Accuracy	94.37%	94.37%	88.70%	86.27%	55.63%	83.90%	87.90%
Precision	0.95	0.95	0.86	0.83	0.56	0.78	0.86
Recall	0.95	0.95	0.97	0.96	1.00	1.00	0.94
F1 Score	0.95	0.95	0.91	0.89	0.71	0.88	0.90
TP	13	13	13	13	14	14	13
FP	1	1	2	3	11	4	2
FN	1	1	0	1	0	0	1
TN	10	10	9	8	0	7	9

Nearest neighbours is one of the most intuitive and basic machine learning algorithms. It calculates the distance (euclidean distance in our case) from the test observation to all the training samples and

predicts the test class to be the same as the closest training observation. An obvious drawback here is that we can be trusting outliers in the training set and severely over-fitting. For these reasons, the learning algorithm is often extended to K-Nearest Neighbours which we will explore later.

Since the basic Nearest Neighbours has no inherent parameters, we don't need to parameter tune and can instead simply run this for each feature and observe the best performers, using the training pipeline provided in the previous section.

Table 2 shows unexpectedly high performance for such a basic algorithm. As illustrated in Figure 22 below, many images in the original 124 image dataset are the same people with very slight variations (e.g. changes in illumination). Consequently, when cross-validation is applied over the entire image set, in collaboration with our pre-processing, the model may encounter near-identical images in both training and testing folds, inflating evaluation metrics for a model that overfits on the training data so heavily. We would suspect that such a model would perform poorly on entirely unseen data, which we will confirm later during detection. With all this in mind, we cannot rely on the results in this table to be accurate to unseen data and going forward into K-Nearest Neighbours we will consider options to make this model less liable to overfitting on the training data and ignore these biased results.

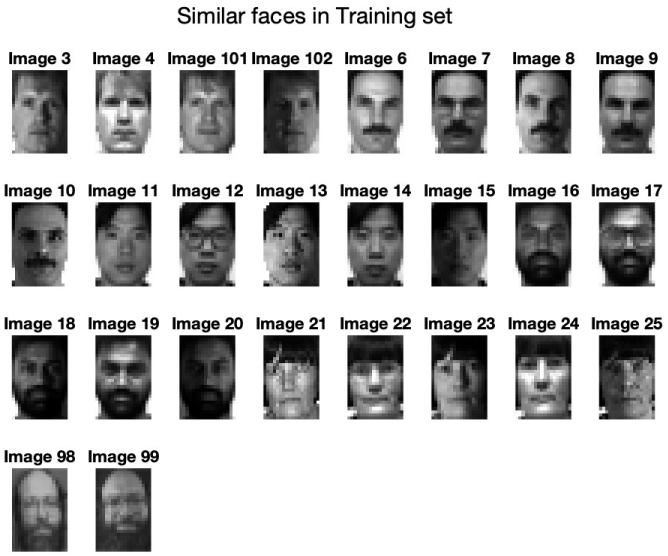


Figure 22: Similar Faces in Dataset

K-Nearest Neighbours (KNN)

KNN extends the nearest neighbour algorithm by predicting the test label based on the majority class from the K-nearest samples. Table 3 presents the results of running KNN with K = $\sqrt{N}$, where N is the number of training images, across different feature types. Similar to nearest neighbours, the raw-pixel features show unexpectedly high performance, likely due to strong correlations between the training and test sets and KNN's susceptibility to overfitting with high-dimensional data.

Table 3: Results Table For K-Nearest Neighbours (K=$\sqrt{N} = 11$)

	RawPix	**PCA**	**Edges**	**EdgesPCA**	**GaborPCA**	**HOG**	**LBP**
Accuracy	92.00%	92.77%	84.67%	84.67%	55.63%	71.73%	87.90%
Precision	0.90	0.91	0.79	0.79	0.56	0.67	0.86
Recall	0.97	0.97	0.98	0.98	1.00	1.00	0.96
F1 Score	0.93	0.94	0.88	0.88	0.71	0.80	0.90
TP	13	13	14	14	14	14	13
FP	2	1	4	4	11	7	2
FN	0	0	0	0	0	0	1
TN	9	10	7	7	0	4	9

To illustrate further, Figure 23 shows the an example of KNN's performance across odd values of K from 1 to 49, revealing a decline in accuracy as K increases - a sign of overfitting on train a training set that is correlated to the test set. Since we cannot trust these results, we will proceed with K = $\sqrt{N}$, a common and reasonable default in the absence of reliable information for unseen data. We will bring forward PCA, LBP and EdgesPCA as our feature extractors for KNN to perform detection with and see how this model performs in practice on unseen data to confirm our suspicions.

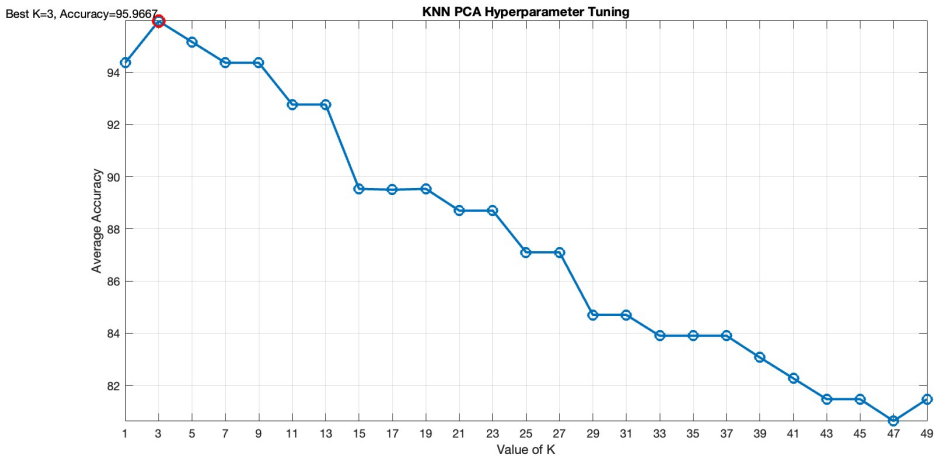


Figure 23: KNN PCA Grid Search

6.3 Support Vector Machine (SVM)

Overview of Model

Support Vector Machines (SVM) are powerful supervised learning models used for both binary and multi-class classification. SVMs work by finding the optimal hyperplane that maximally separates the data points of different classes in a high-dimensional space. This hyperplane is determined by the support vectors (data points closest to the decision boundary) which makes the model robust to outliers. SVM can handle non-linear relationships by applying kernel functions, which transform the input data into a higher dimension where a linear separator can be found. We predict SVM to be one of our best performers in the face detection tasks due to its effectiveness in handling high-dimensional data, small training datasets and its flexibility through kernel selection, providing a strong benchmark for comparison with other models.

Parameter Tuning in SVM

One drawback of SVM is that it needs extremely careful tuning and is hyper sensitive to change in parameter settings and kernel functions. With this is mind we can't take a similar approach to other models where we choose a "sensible" parameter value to begin with to distinguish the best feature descriptors. Instead we follow a procedure laid out by Chih-Wei Hsu in "A Practical Guide to Support Vector Classification" [6], where they suggest the following:

1. Conduct simple scaling on the data
2. Consider the RBF (Gaussian) kernel
3. Use cross-validation and loose-grid search to find the best parameter C and γ
4. Repeat (3) with a finer grid search to get exact values for C and γ
5. Use the best parameter C and γ to train on the whole training set

We will explain in the next paragraph what the hyper-parameter's C and γ represent and carry out the above steps over the Radial Basis Function Kernel as suggested. Considering this is a computationally expensive and time consuming task, we will perform this on a subsection of our feature descriptors which we consider to be the most promising based on previous model results. Should we find later on the model is performing sub-optimally or worse than expected in the detection task, we can revisit this and try other feature descriptors. For now we will tune our model on Canny Edges, HOG and LBP.

Gaussian (RBF) Kernel

The Gaussian variation of the Radial-basis function kernel found in the SVM-KM package in our code base can be described mathematically as follows:

$$K(\mathbf{x}, \mathbf{x}^{\text{sup}}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}^{\text{sup}}\|^2}{2\gamma^2}\right), \gamma > 0 \tag{1}$$

Here $\mathbf{x}$ is the input feature vector (the data point you want to classify), $\mathbf{x}^{\text{sup}}$ is a support vector from the training set that helps define the decision boundary. The kernel function itself $K(\mathbf{x}, \mathbf{x}^{\text{sup}})$ measures the similarity between $\mathbf{x}$ and each support vector $\mathbf{x}^{\text{sup}}$. The squared euclidean distance between $\mathbf{x}, \mathbf{x}^{\text{sup}}$ represents the similarity between the support vector and test data point, with smaller distances representing close similarity. Combined with γ which serves the purpose of controlling the influence of a single support vector on the decision boundary. Small γ makes the kernel function more "spread out", meaning that each support vector has a broader influence over the decision boundary which could potentially lead to underfitting. A large value γ makes the kernel function more "localised", meaning that each support vector only influences points close to it, which lead to more complex decision boundaries and possible overfitting. The hyper-parameter C is not used directly in the kernel function calculation but instead during training and serves the purpose of regularisation and controlling the complexity of the model. Specifically, it dictates how much penalty is applied for misclassification errors during training. A high C value gives a model the freedom to focus on minimising training errors, possibly at the expense of overfitting, while a low C allows for more errors on the training data but encourages a simpler model that might generalise better.

Hyper-parameter Optimisation

We begin by running a loose-grid search over the range of hyper-parameters $C = (2^{-5}, 2^{-3}, \ldots, 2^{15})$ and $\gamma = (2^{-5}, 2^{-3}, \ldots, 2^{7})$ with our proposed feature descriptors. We obtained the following results displayed in Table 4 below:

Table 4: SVM Gaussian Kernel HPO Loose GS

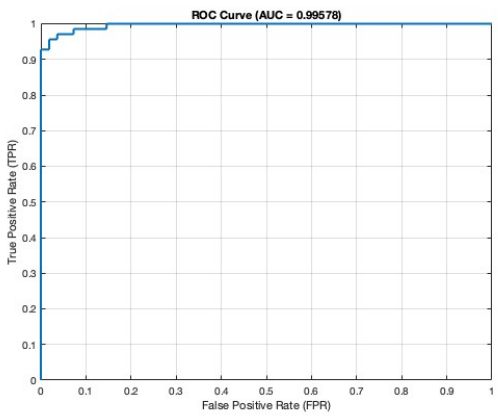
	HOG	**LBP**	**EdgesPCA**
Best γ	0.5	8	16
Best C	8	64	8
Accuracy	96.77%	95.20%	79.73%

After a finer grid search around values closer to the best values we obtained in table 4 we obtained the finalised best performing hyper-parameters for each feature. These results are showcased below in Table 5. We will bring these finalised models trained on the entire augmented image set into the detection task.

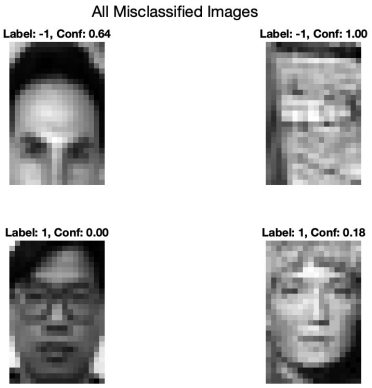
Table 5: SVM Gaussian Kernel HPO Fine GS

	HOG	**LBP**	**EdgesPCA**
Best γ	0.5	5	13
Best C	8	25	8
Accuracy	96.77%	95.20%	80.60%
Precision	0.97	0.95	0.77
Recall	0.97	0.97	0.93
F1-Score	0.97	0.95	0.84
TP	13	13	13
TN	11	10	7
FP	0	1	4
FN	1	0	1

With HOG and LBP performing very well in collaboration with the Gaussian kernel we will take a deeper look into these models below.



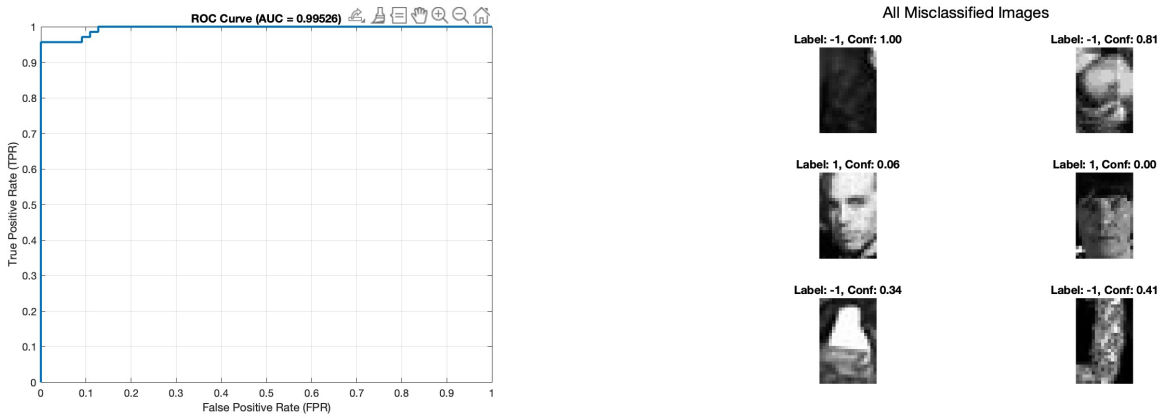
(a) HOG Roc Curve



(b) HOG Misclassifications

Figure 24: SVM Hog Model Evaluation

HOG achieves an AUC of 0.99 with only four misclassifications. One of these errors involves a face with glasses, classified incorrectly with high confidence. This is likely because the glasses obscure key facial features, disrupting HOG's reliance on textures and edges. Interestingly, the other misclassified face is unusual because it bears a strong resemblance to a face, making the incorrect classification somewhat unexpected. The non face misclassification involves a image of a half-face, likely due to the presence of an outline resembling the majority of a face. This creates textures and edges similar to those of an actual face. The final misclassified non-face image contains features that have high variance in textures and edges however the 1.00 confidence is quite strange to see here.



(a) LBP Roc Curve

(b) LBP Misclassifications

Figure 25: SVM Hog Model Evaluation

LBP performed slightly worse with 6 misclassifications, however, it still maintained an exceptionally high AUC of 0.99. These misclassification are slightly less explicable than HOG's. The non frontal facing face is the only obvious misclassification as we are training on frontal facing images. The other misclassification are harder to justify. Keeping in mind the model only misclassifies 6 images during testing, we still expect this feature to perform well and will explore it further in the detection task.

Polynomial Kernel

Another common choice for a kernel is the polynomial kernel. For simple face detection, an SVM with a Gaussian RBF kernel is typically expected to perform better due to its flexibility and ability to model complex decision boundaries. However, since we have a small dataset sometimes a more simplistic approach such as a Polynomial kernel might also be effective. In the case of our codebase we have the following polynomial kernel function implemented, where the polynomial constant value is fixed at 1.

$$K(\mathbf{x}, \mathbf{x}^{\text{sup}}) = (\mathbf{x} \cdot \mathbf{x}^{\text{sup}} + 1)^{degree} \tag{2}$$

With this in mind we briefly explored some parameters against the polynomial kernel on a loose grid search of polynomial degrees between 1 and 5 (we kept the C value fixed at 8).

The results outlined in Table 6 show some similar values when compared to the gaussian RBF kernel for EdgesPCA, HOG and LBP. So, should the gaussian kernel underperform in detection we can explore the polynomial kernel further and fine tune this later on.

Table 6: Results Table For SVM after HPO for Polynomial Degree

	Degree 3	Degree 2	Degree 2	Degree 1	N/A	Degree 5	Degree 3
	RawPix	**PCA**	**Edges**	**EdgesPCA**	**GaborPCA**	**HOG**	**LBP**
Accuracy	95.13%	90.33%	70.00%	78.13%	55.63%	96.00%	95.20%
Precision	0.96	0.58	0.79	0.76	0.56	0.97	0.95
Recall	0.96	0.97	0.95	0.88	1.00	0.96	0.97
F1 Score	0.96	0.97	0.86	0.82	0.71	0.96	0.96
TP	13	13	13	12	14	13	13
TN	10	9	7	7	0	11	10
FP	1	2	4	4	11	0	1
FN	1	0	1	2	0	1	0

6.4 Logistic Regression

Overview of Model

Logistic Regression is a basic supervised machine learning model used for binary classification. It predicts the probability of an input belonging to a particular class using a sigmoid activation function, which maps linear combinations of features to probabilities between 0 and 1. This probability between 0 and 1 is compared to threshold (usually 0.5) which then classifies it to the two categories. We choose this model due to its basic nature and easy to implementability using default parameters as there is no inherently obvious parameters to tune. Using it to stand as sort of a baseline to judge our other models on.

Table 7: Results Table For Logistic Regression

	RawPix	**PCA**	**Edges**	**EdgesPCA**	**GaborPCA**	**HOG**	**LBP**
Accuracy	63.33%	83.33%	70.00%	70.00%	66.67%	90.00%	73.33%
Precision	0.80	0.89	0.79	0.76	0.67	0.90	1.00
Recall	0.60	0.85	0.75	0.80	1.00	0.95	0.60
F1 Score	0.69	0.87	0.77	0.78	0.80	0.93	0.75
TP	12	17	15	16	20	19	12
TN	7	8	6	5	0	8	10
FP	3	2	4	5	10	2	0
FN	8	3	5	4	0	1	8

Underperforming Feature Extraction Techniques

As shown in Table 7, most feature extraction techniques perform poorly in this binary classification

task when paired with logistic regression, including raw pixel values, Edges, EdgesPCA, GaborPCA, and LBP, due to several key limitations.

Raw pixel values face significant challenges due to high dimensionality, making the model prone to overfitting. Additionally, the lack of meaningful feature extraction leaves the model sensitive to pixel intensity variability caused by lighting, pose, or expression. This is evident in the results, where the model correctly classified only 12 faces and 7 non-faces, with many faces misclassified as non-faces due to the high variability among face images.

Edge-based features achieved slightly better performance, with an accuracy of 70.00% and an F1-score of 0.77, but still failed to capture the structural and contextual information necessary for reliable classification. PCA applied to edges yielded nearly identical results, indicating that the issue lies in the limited discriminative power of the edge features, not dimensionality.

Gabor features with PCA achieved an accuracy of 66.67% and an F1-score of 0.80 but exhibited a recall of 1.00, meaning all inputs were classified as "faces". The feature space created by Gabor filters and PCA aligns well with facial patterns but poorly represents non-faces, likely due to the dataset's class imbalance (20 faces and 10 non-faces), which made non-faces appear as outliers.

LBP achieved an accuracy of 73.33% and an F1-score of 0.75, with the model misclassifying 8 out of 20 faces but correctly identifying all non-faces. However, LBP's focus on local texture patterns rather than global structural patterns explains its weaker performance on faces, as facial information is often embedded in spatial relationships between features like the eyes, nose, and mouth. Additionally, the small image size may cause important details to be lost, with features like the eyes blending into nearby textures, reducing the model's effectiveness.

PCA achieved an accuracy of 83.33% and an F1-score of 0.87, demonstrating its strength in both face and non-face classification. Specifically, PCA correctly classified 17 out of 20 faces and 8 out of 10 non-faces, showcasing its ability to distinguish between the two classes effectively. This success can be attributed to PCA's ability to reduce data dimensionality while retaining the most relevant features, eliminating noise and redundancy. By maximising variance in the data, PCA emphasises global patterns such as facial structure, while also creating a feature space where the greater variability in non-faces is better represented. This transformation makes the data more linearly separable and reduces overfitting, allowing logistic regression to generalise well and achieve strong classification performance.

Histogram of Oriented Gradients (HOG)

HOG emerged as the best overall performer, with an accuracy of 90.00% and an F1-score of 0.93, reflecting its balanced ability to classify both faces and non-faces effectively. The model correctly

classified 19 out of 20 faces and 8 out of 10 non-faces, demonstrating robustness across both classes.

The superior performance of HOG lies in its ability to extract edge-like orientations and structural patterns from images. Faces exhibit predictable gradient patterns due to their consistent structure, such as the alignment of facial features (eyes, nose, mouth), which HOG captures effectively. In contrast, non-faces, often characterised by smoother or irregular textures, produce less consistent gradient patterns, making them easier to differentiate from faces.

Furthermore, HOG's design inherently encodes structural and gradient-based patterns in a way that is linearly separable. This aligns well with logistic regression, which performs best when the feature space allows for linear decision boundaries. By dividing the image into localised cells and computing histograms of gradient orientations, HOG captures both local and global information, resulting in features that are discriminative and robust to variations in lighting, pose, and background.

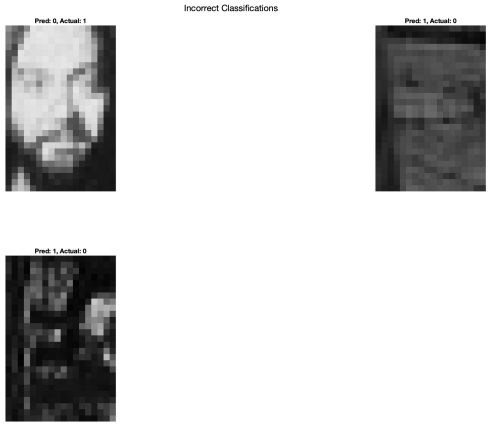


Figure 26: Logistic Regression Misclassified Images

The misclassified face in Figure 26 is likely due to its faint facial features, which lack the strong gradients and contrasts that HOG relies on to extract meaningful patterns. HOG focuses on detecting local gradients, such as edges and contours, but when facial features are low in contrast, it struggles to capture the necessary details, leading the model to misinterpret the face as a non-face. Similarly, the non-face images are very dark and exhibit minimal variation in texture or structure. These dark areas may still have some edge patterns that resemble facial components, such as shadows or contours, confusing the HOG feature extraction and the classifier. The lack of variability and meaningful gradients in these images contributes to their misclassification as faces, as the model struggles to distinguish between ambiguous patterns, resulting in misclassification.

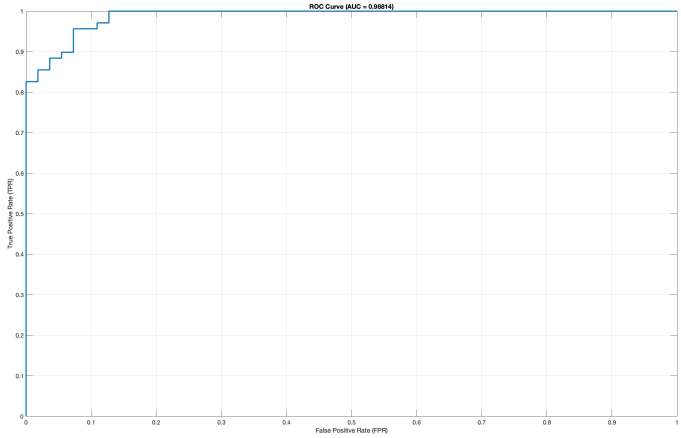


Figure 27: Logistic Regression HOG ROC Curve

This ROC curve in Figure 27 demonstrates the effectiveness of the logistic regression model when paired with HOG features, as reflected in the exceptionally high AUC value of 0.98814. This indicates that the model has excellent discriminative power, effectively distinguishing between faces and non-faces across various thresholds. The steep rise in the curve and its proximity to the top-left corner highlight the model's ability to achieve a high true positive rate (sensitivity) while maintaining a low false positive rate. These results are significant because they validate the combination of logistic regression and HOG features as a robust approach for classification tasks where local texture and gradient-based information play a key role. This suggests that the model generalises well and can reliably identify faces while minimising misclassification errors. Which is why we will bring this model forward to the next stage of face detection.

7 Detection

7.1 Sliding Window Algorithm

The Sliding Window Algorithm represents a fundamental approach to face detection. At its core, the algorithm works by moving a fixed-size detection window (27x18 pixels in our implementation) across an image, analysing each region as a binary classification problem: face or non-face. The detection window moves across the image using a predetermined step size. Starting from the top-left corner (1,1), it scans horizontally along the x-axis until reaching the image's right edge. Once it reaches the

edge, the window returns to the leftmost position (x=1) and moves down one pixel on the y-axis (y=2) to begin scanning the next row. This scanning pattern ensures every possible location in the image is evaluated for the presence of a face.

To account for faces of varying sizes within an image, the algorithm can process multiple scales of the input image or detection window, enabling detection of faces at different distances from the camera. While this comprehensive approach is computationally intensive due to the large number of windows evaluated, its thoroughness and reliability make it particularly effective for face detection tasks. The algorithm's performance can also be further enhanced through post-processing steps, such as confidence thresholding, which filters out weak detections below a predetermined confidence score, and non-maximum suppression, which eliminates redundant overlapping detections.

The following sections evaluate the key variations of our sliding window implementation: horizontal scanning & vertical scanning, single-scale detection, multi-scale detection, and the integration of confidence thresholding and non-maximum suppression.

7.2 Horizontal & Vertical Scanning

In the implementation of our sliding window detection algorithm, we deliberately opted for horizontal-only scanning rather than bidirectional scanning, a decision driven by both computational efficiency considerations and the effectiveness of unidirectional scanning for our detection task. (If the below gif is not rendering in your pdf reader you can view it here: https://imgur.com/a/k5F5C4O)

The time complexity of horizontal scanning alone is O($n \cdot m$), where n is the number of rows and m is the number of columns in the image. Adding vertical scanning would double the computational cost to process the same image area and considering we are using a step size of [1,1] the image is already being thoroughly scanned. In practice we saw little to no difference in results applying both so we stuck to just horizontal scanning with a step size of one for our final detection algorithm.

7.3 Single Scale Detection

Our initial implementation utilised a single-scale detection approach with a fixed bounding box of 27x18 pixels. The algorithm systematically scans the image with a step size of [1,1], evaluating each window and predicting a confidence score. Windows with confidence scores exceeding a predefined threshold (set to 0.65 in our case) are considered positive detections. These windows are then used to draw bounding boxes on the original image, highlighting the detected regions. While this comprehensive scanning ensures no potential face region is missed, it introduces a significant challenge: multiple detections of the same face occur due to the dense sampling pattern. This overlap in detection regions is a natural consequence of the small step size, as adjacent windows capture largely identical portions of the image, leading to redundant positive classifications of the same facial features.

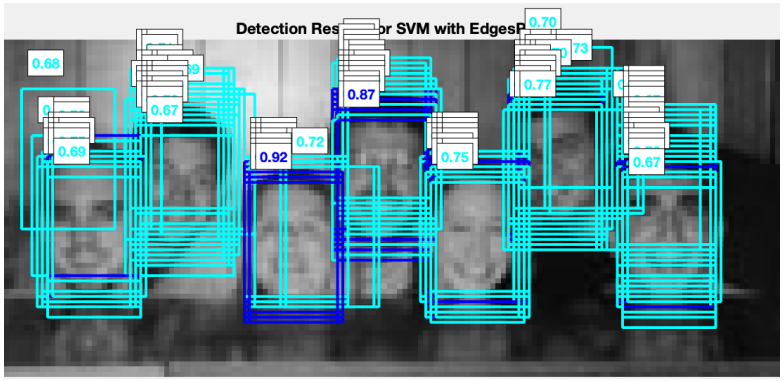


Figure 28: Single scale detection no NMS

As seen in Figure 28 the model identified 157 potential face regions exceeding our confidence threshold, yet without an effective mechanism for filtering redundant detections, the results are suboptimal. This high number of overlapping detections demonstrates the clear need for a more sophisticated post-processing strategy. A common technique used for this is non-maxima suppression.

7.4 Non-maxima Supression (NMS)

Our NMS implementation evaluates pairs of bounding boxes by calculating their intersection area and comparing it against the area of the smaller box to determine the overlap ratio. When this ratio exceeds our pre-defined threshold (0.05 in our case), the algorithm retains only the detection with the higher confidence score, effectively pruning weaker, overlapping detections.

This approach proves highly effective at consolidating multiple detections of the same face into a single, optimal bounding box.

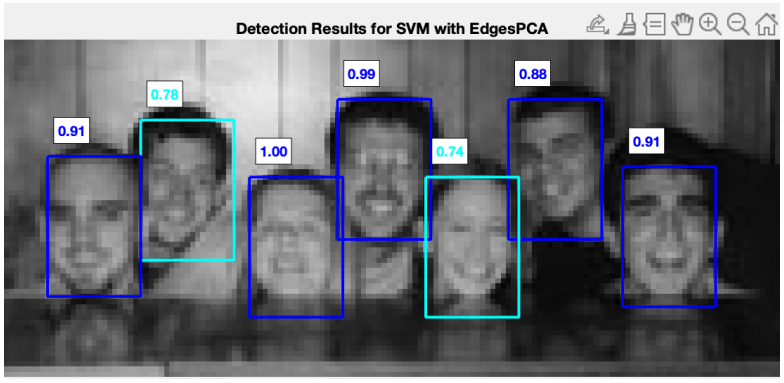


Figure 29: Single Sale Detection With NMS

As seen in Figure 29, where our basic implementation previously identified 157 potential faces above the confidence threshold, NMS successfully filtered these to retain only the strongest, non-overlapping detections, substantially improving the precision of our sliding window.

7.5 Multi-Scale Detection

Intuitively Multi-scale detection emerges as the natural progression in our sliding window implementation, it extends the traditional sliding window to analyse images at multiple scales. Our implementation is effective because it enables the detection of faces at various depths and sizes within an image by systematically resizing the input image while maintaining a fixed size window. By applying non-maxima suppression both at individual scales and across final detections, we found the best results were achieved.

Figure 30: Detecting scales [0.9,1,1.1]

As seen in Figure 30 above here are the detection results when running multi-scale detection at scales [0.9, 1, 1.1], to visualise the bounding box sizes non maxima suppression is only being applied per scale.

Using this approach has improved model performance particularly in the more complex images that include faces at various distances and sizes, while this technique showed significant improvements in scenarios with faces at different depths and scales, there were no notable performance differences when processing images where all faces maintained consistent depth and size relative to the sensor. We found the optimal approach in practice was to use single scale on the first 3 images, as multi-scale increased the time complexity without improving results, and scaling the original image to [1, 1.05, 1.1] on image 4 to be the most effective approach. Of course in practice on entirely unseen data we would not know this in advance and we would suggest to simply run the model at several scales if computational performance is not a worry as this will yield the optimal results.

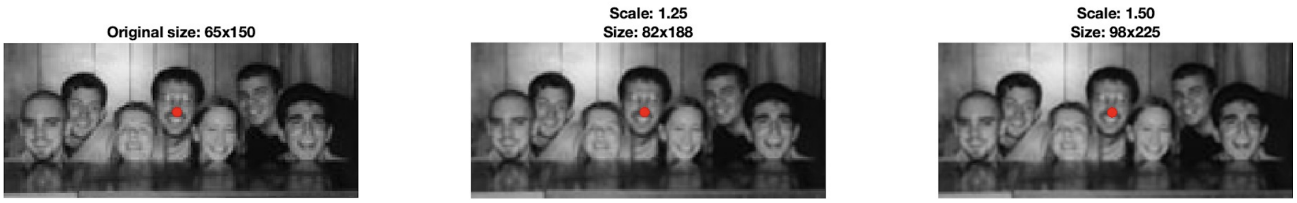


Figure 31: Plotting point at different scales to show operational multi-scaling

Due to Matlab's subplot constraints in maintaining uniform figure dimensions, the actual size variations between images cannot be visually represented. Reference measurements for each image's dimensions are provided in the annotations above their respective subplots.

7.6 Model Evaluation Metrics

Ground Truths and Intersection over Union

The foundation of our evaluation lies in ground truth annotations, which are manually created bounding boxes that precisely identify face locations in our detection image test set. To quantify the accuracy of our detections, we utilise the Intersection over Union (IoU) metric, which measures the overlap between predicted bounding boxes and ground truth boxes. The (IoU) is calculated as the area of intersection divided by the area of union between the predicted and ground truth boxes, providing a score between 0 and 1, where 1 indicates perfect overlap.

Classification Metrics

Our evaluation system tracks several key error types and success metrics:

True Positives (TP): True Positives represent detected faces that correctly overlap with ground truth boxes above our (IoU) threshold of 0.5. Each detection is matched to one ground truth box, prioritising matches with higher confidence scores

False Positives (FP): False positives occur when our system reports a detection that either doesn't overlap sufficiently with any ground truth box (IoU < 0.5) or overlaps with a ground truth box that has already been matched to a higher confidence detection.

False Negatives (FN): False Negatives represent ground truth faces that our system failed to detect, calculated as the number of ground truth boxes that remain unmatched to any detection.

True negatives have been excluded from our evaluation metrics to ensure our performance measurements accurately reflect the detector's core capability of identifying faces. Consider that in our smallest test image, the sliding window evaluates 7,910 potential face locations, yet only 7 of these contain actual faces. This means 7,903 windows represent true negatives. Including true negatives in our metrics would bias our results towards the models ability to reject non-faces rather than its core function of accurate face detection.

By focusing instead on true positives, false positives and false negatives, we capture meaningful performance metrics that accurately represent our detector's capabilities.

Performance Metrics

From the basic classifications above, we derive three crucial performance metrics:

Accuracy = TP/(TP+FP+FN)

Accuracy provides the overall measure of system performance, considering both successful detections and both type errors.

Precision: TP/(TP+FP)

Precision measures the proportion of our detections that are correct, helping us understand how reliable our positive predictions are.

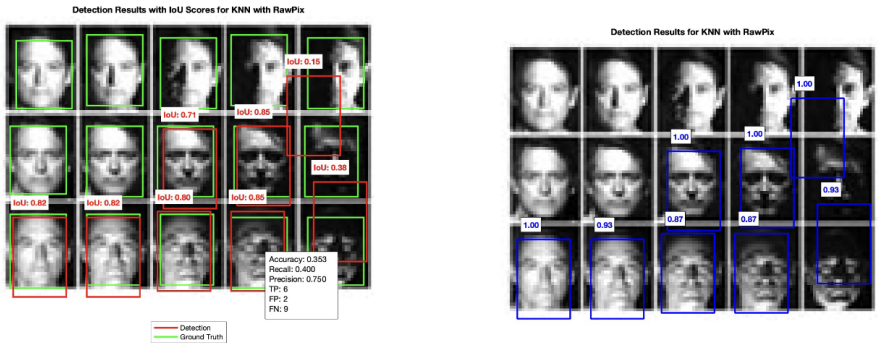
Recall: TP/(TP+FN)

Recall indicates the proportion of actual faces that our system successfully detected, helping us understand our systems performance in finding faces.

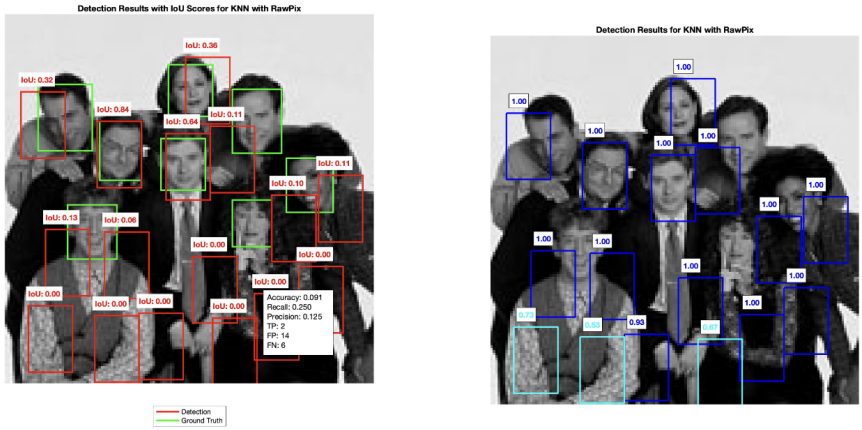
7.7 Underperforming Models

Nearest Neighbours & K-Nearest Neighbours

To begin with we will showcase what a weak classifier looks like in our case. Here we show the performance of KNN trained on the entire augmented image set (910 images), using a value of $K = \sqrt{N} = \sqrt{910} = 30$.



(a) KNN Detection Results Image 2



(b) KNN Detection Results Image 3

Figure 32: KNN Detection Results Image 2 & Image 3

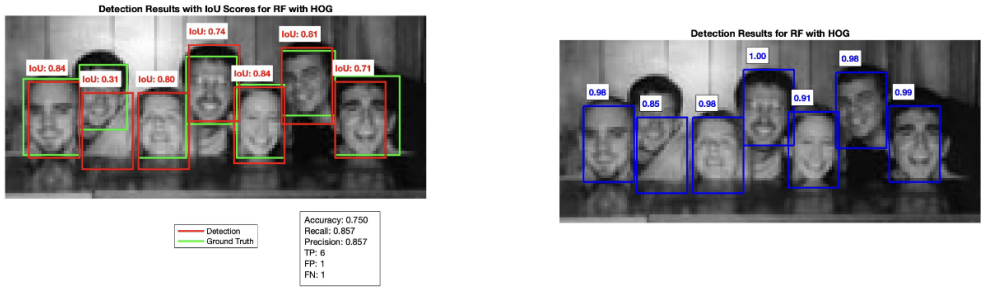
As shown in Figure 32, the inflated cross-validation results for KNN have been debunked. When applied to the detection task, KNN significantly underperforms compared to its testing phase results, as anticipated. Recall that during testing, KNN with raw pixel features achieved an accuracy of 92%,

which was unexpected. We previously discussed possible reasons for this and expressed concerns about its ability to generalise to unseen detection images.

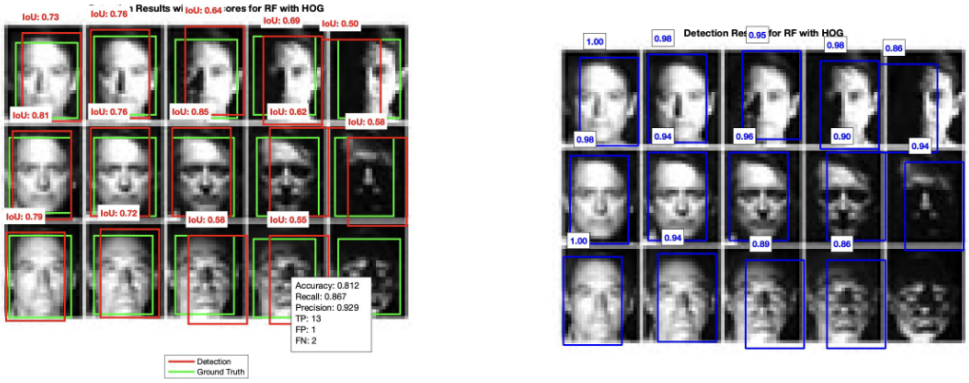
KNN's struggles with the curse of dimensionality are particularly pronounced when relying on raw pixel features, as it fails to capture meaningful facial patterns. This limitation arises because Euclidean distance, the core metric of KNN, measures only pixel-level proximity rather than the deeper semantic similarities between faces. As illustrated in Figure 32(b), this shortcoming results in KNN achieving a mere 0.091 accuracy, with only 2 true positives and 14 false positives, highlighting its inability to generalise effectively.

Performance worsened further when we reduced the K value or employed a simple nearest-neighbour approach as it begins to overfit on the training data, as discussed earlier. While slight improvements were observed when combining KNN with other feature extractors to mitigate its reliance on Euclidean distance on raw pixel intensities, the overall results remained underwhelming. Consequently, we shift our focus toward more sophisticated models that better align with the demands of face detection.

Random Forest



(a) Random Forest HOG Detection Results Image 1

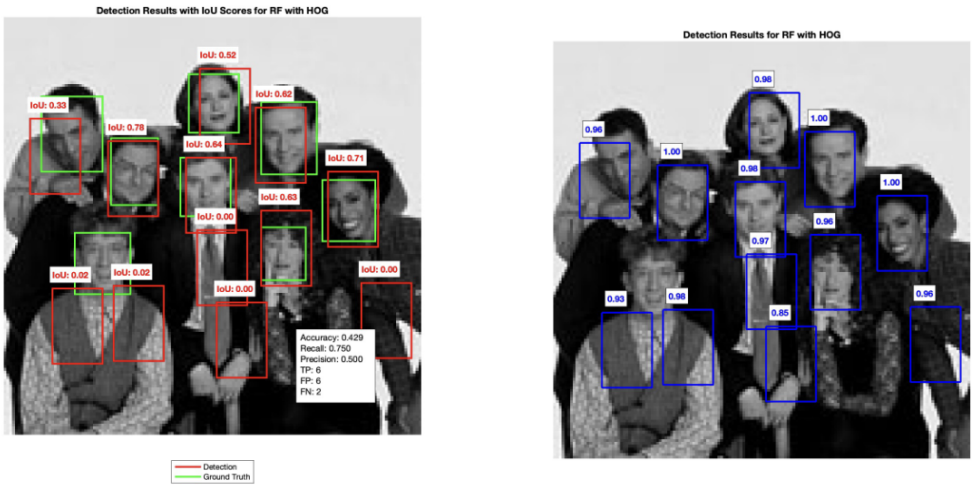


(b) Random Forest HOG Detection Results Image 2

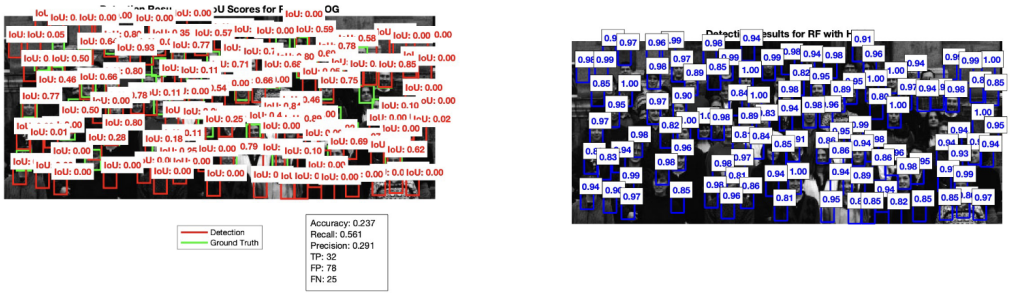
Figure 33: Random Forest HOG Detection Results Image 1 & Image 2

The Random Forest model combined with HOG features was the best-performing model during testing, so we evaluated its detection performance on the images shown above. In the 22 faces in Figure 33, the model correctly classified 19 of them with only predicting an additional 2 non-faces as faces. Achieving accuracies of 75% and 81.2%, across image 1 and 2 respectively.

This model appears to be performing relatively well on the simpler images. However, the model is computationally intensive. For instance, single-scale detection on image 1 required 255 seconds. Given these results, while the model shows promise in these simpler scenarios, its performance may not inspire confidence when applied to more complex images, where the diversity of faces, poses, and the image size increasing are all worries here.



(a) Random Forest HOG Detection Results Image 3



(b) Random Forest HOG Detection Results Image 4

Figure 34: Random Forest HOG Detection Results Image 3 & Image 4

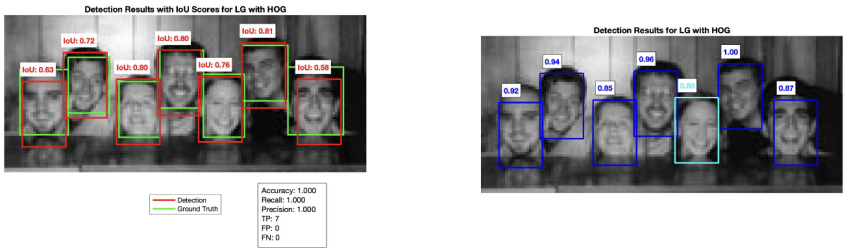
With these two final images, it is evident that the model struggles with false positives. For instance, in Figure 34(a), the model achieves a precision of 0.5, indicating that for every correctly classified face, there is another incorrectly classified as a face. These false positives often occur in regions with prominent edges or varying texture gradients. One notable example is a partially classified face at an angle, suggesting that the model requires more diverse training data, including images with varied angles, poses, and facial expressions, to improve accuracy in such scenarios.

In Figure 34(b), this issue is even more pronounced, with a precision of just 0.291. This highlights the model's significant difficulty in detecting faces accurately and reinforces its limitations. Surprisingly, ensemble methods like Random Forests are typically robust against false positives due to the voting mechanism among trees. However, this underperformance is likely a result of the small and limited dataset used for training, which lacked the diversity necessary to generalise effectively on the unseen

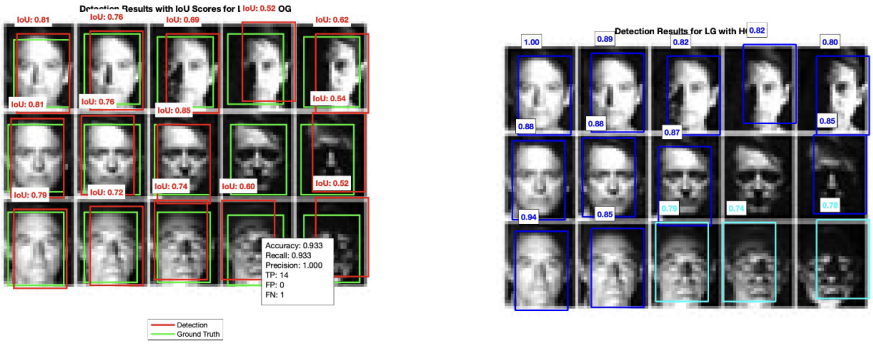
data.

Furthermore, we were unable to run this model at multiple scales due to its computational inefficiency. Even at a single scale, the model took 7,329 seconds (over 2 hours) to complete. When compared to our other algorithms, this is on average 20x more computationally expensive. Given these underwhelming results, it is clear that this model is impractical for face detection tasks when trained on a small data set, as it is both computationally expensive and ineffective.

Logistic Regression



(a) Logistic Regression HOG Detection Results Image 1

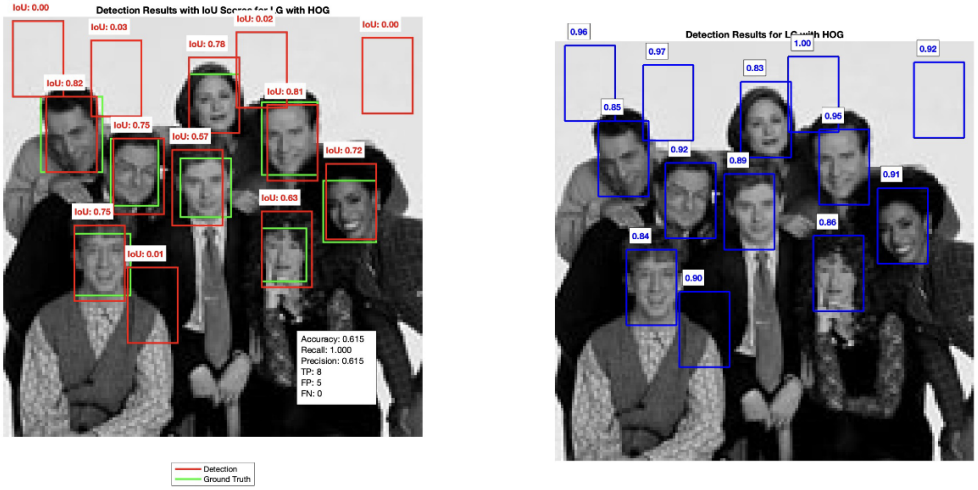


(b) Logistic Regression Detection HOG Results Image 2

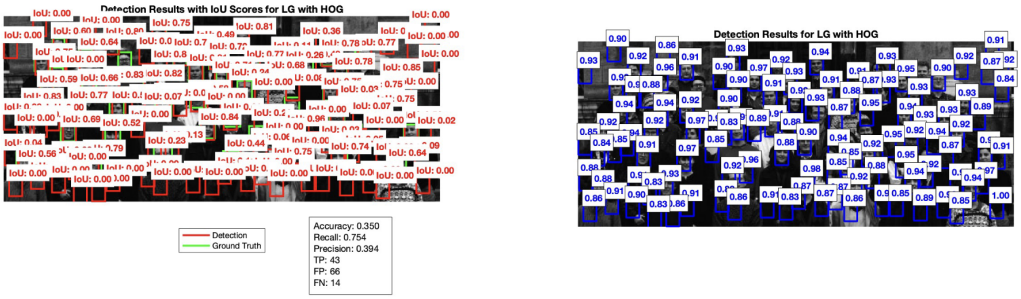
Figure 35: Logistic Regression HOG Detection Results Image 1 & Image 2

Logistic regression combined with HOG features was our best performing logistic regression model. Which aligns with our findings during training and testing. As illustrated in Figure 35, it had a total of 21 True Positives, 1 False Negative and 0 False positives. This 100% level of precision combined with high accuracy is an ideal model for the task as it is avoiding over-predicting faces, which has been a common trend with other models. Logistic Regression performing this well on entirely unseen data

surprised us, due to the simple decision boundary of Logistic Regression and the complexity. We had previously planned to briefly touch on this model, however, because of these promising results we will test it on the final two images and assess it's performance there.



(a) Logistic Regression HOG Detection Results Image 3



(b) Logistic Regression HOG Detection Results Image 4

Figure 36: Logistic Regression HOG Detection Results Image 3 & Image 4

A surprising trend emerges on the final two images. The model begins to predict significantly more false positives, with some of them being hard to justify. For example, in Figure 36(a), 4 of the false positives are on a plain background. This may suggest that the model favours low frequency edge orientations to be associated with faces however this is not what one would expect. Another possible reason could be due to noise causing HOG to extract high-frequency edges that are in fact redundant. The results in Figure 36(b) are not as surprising as this image is more complex with different poses and overlapping objects (people). However, the number of false positives is extremely high and does not

align with the high precision levels we say in the initial two images. It is difficult to give a concrete reason as to why this is the case other than the complexity and size of the image increasing (more chances to predict wrong) and the simplicity of the logistic regression decision boundary struggling to handle these more difficult cases.

7.8 Best Performing Model - SVM

In our analysis of model performance, Support Vector Machine (SVM) demonstrated superior results, aligning with our expectations and results from training and testing. Our evaluation focuses on two optimised SVM configurations: one employing a Gaussian kernel and another utilising a Polynomial kernel. After exploring several combinations of feature extractors and kernels, we found leveraging edge detection combined with PCA provided the best result for the polynomial kernel. While simply applying PCA to raw-pixel values performed the best in conjunction with the Gaussian kernel.

Gaussian Kernel

As mentioned above we found the best feature extractor with the Gaussian kernel was a simplistic approach of applying PCA on raw pixel values to reduce the dimensionality. Previously we saw in testing that we had good performance with HOG, LBP, EdgesPCA and the Gaussian kernel, the detection for each of these features was still quite good however we felt we could achieve better. With this in mind we then ran our HPO grid search on the PCA model and rather surprisingly found that the optimal a γ value was very high at 1000 and the optimal C value was 25. We trained our model with these hyper-parameters on the entire augmented training set and achieved the following results in detection:

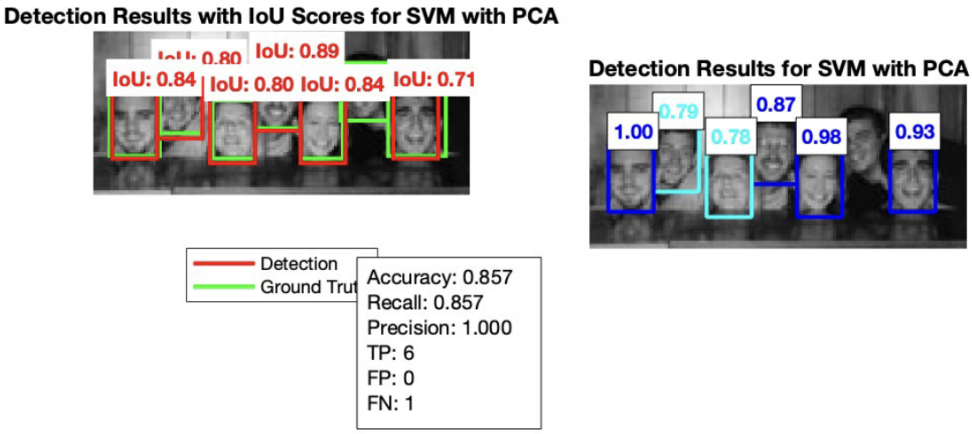


Figure 37: PCA Gaussian Image 1

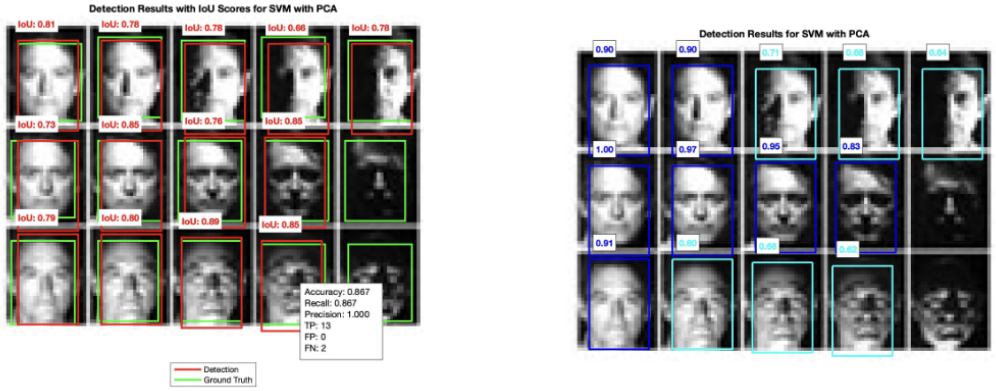


Figure 38: PCA Gaussian Image 2

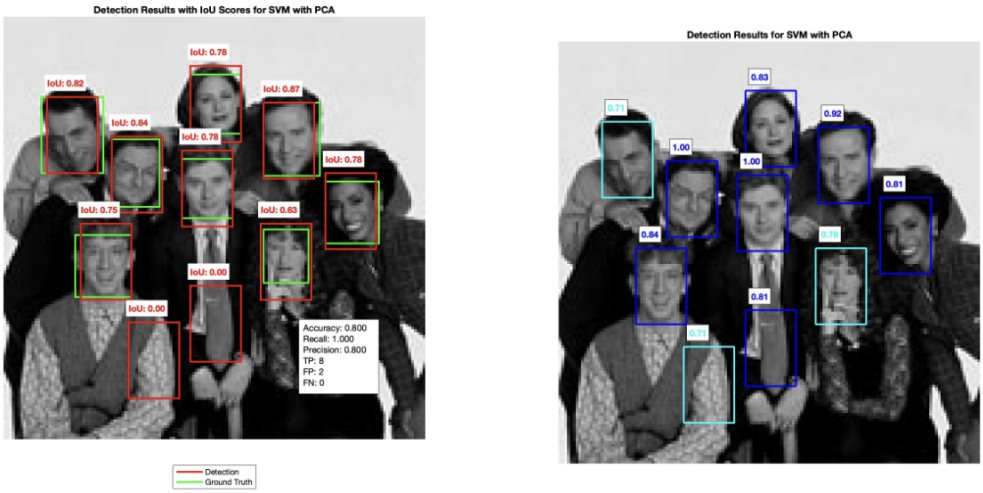


Figure 39: PCA Gaussian Image 3

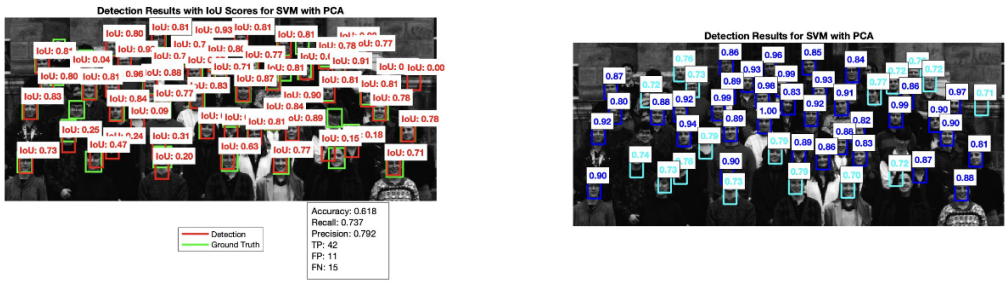


Figure 40: PCA Gaussian Image 4

Table 8: SVM Gaussian Performance Across all Detection Images

Model	TP	FP	FN	Accuracy	Precision	Recall
SVM Gaussian PCA	69	13	18	61.76%	76.36%	73.68%

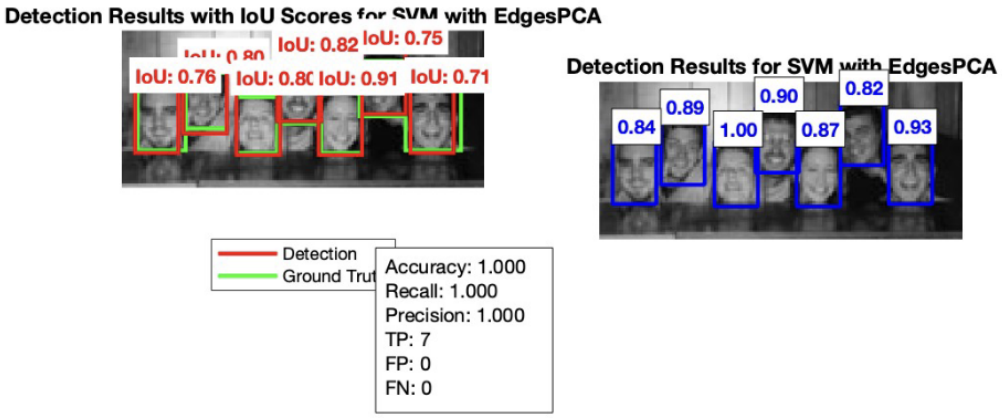
As shown in Figures 37-40, this model outperforms all previous models discussed in this report, delivering above average metrics. However, it still falls short of the optimal performance we aim for. We will not analyse this model's performance in detail, as a better alternative emerged using a polynomial kernel, which will be highlighted in the next section.

We believe the superior performance of the polynomial kernel over the Gaussian kernel can be attributed to two key factors. First, tuning a Gaussian kernel is significantly more complex compared to the relatively straightforward adjustment of the polynomial degree. Second, this challenge is compounded by the limitations of our training and testing sets, which may not provide reliable parameter tuning or realistic results for truly unseen data. We previously discussed these concerns about the reliability of the training and testing sets in our KNN model evaluation, where we observed unusually high performance with low k-values and raw pixel features; results that would likely not generalise well to new data. We believe this problem fed into our HPO of Gaussian kernels.

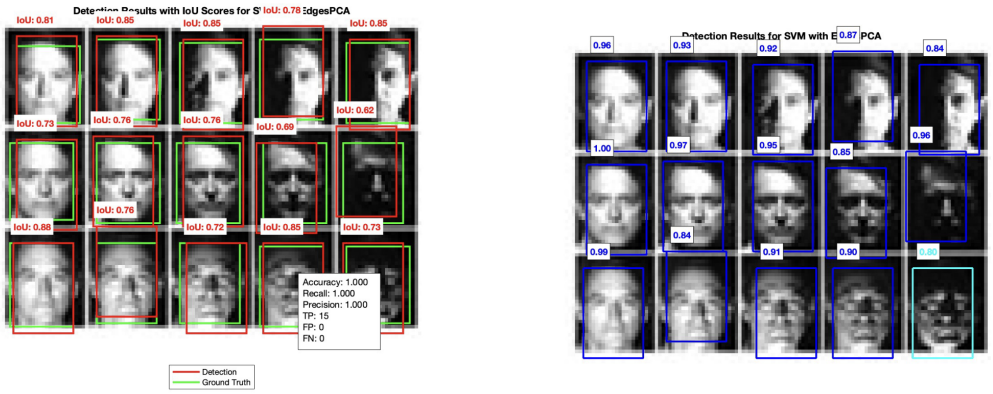
Polynomial Kernel

As mentioned, we found leveraging edge detection combined with PCA provided the best result for the polynomial kernel. We applied a loose-grid search with some further fine tuning and found that a polynomial degree of 2 with a C value of 8 were the optimal hyper-parameters.

Detection on the first and second image demonstrate perfect performance. With an accuracy, recall, and precision of 1.0, successfully detecting all faces with no false positives or false negatives and the Intersection over Union (IoU) scores range from 0.62 - 0.91, indicating accurate detection bounding box placements.



(a) EdgesPCA Polynomial Image 1



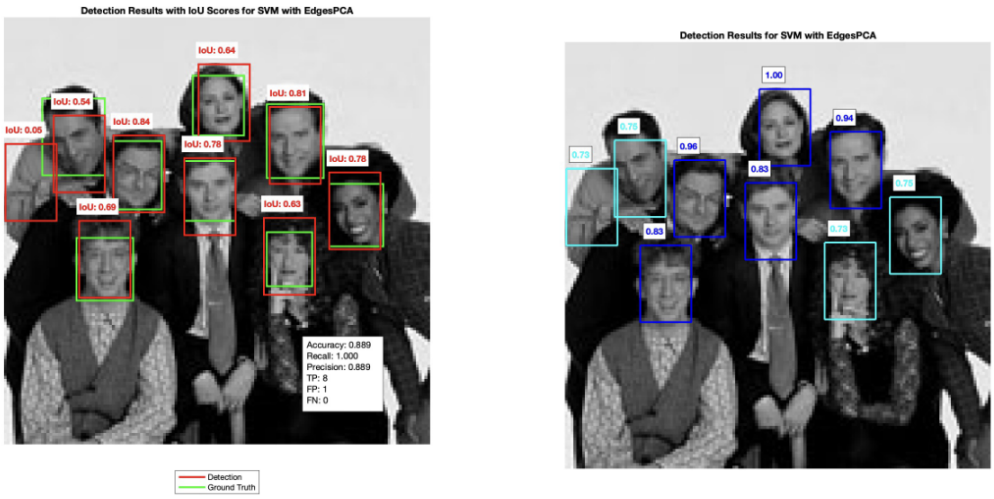
(b) EdgesPCA Polynomial Image 2

Figure 41: SVM Detection Results Image 1 & Image 2

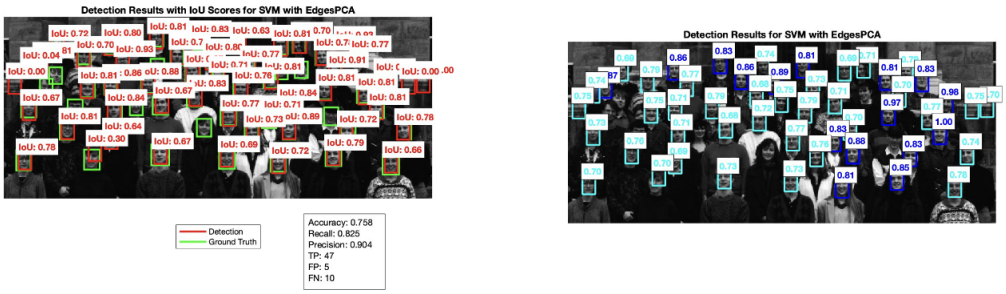
The model demonstrates excellent performance in image one, where faces appear under ideal conditions with consistent scale, orientation and uniform lighting. Even more impressive is its sustained accuracy in image two, where faces vary in size and illumination. This adaptability indicates that our Edges/PCA features and pre-processing techniques combined with SVM classification have effectively learned to handle such variations, rather than relying on consistent presentation of faces.

The simplicity of these detection tasks for our SVM model is primarily due to the controlled nature of these test images. With consistent face sizes, limited pose variations and clear foreground-background separation. These conditions combined with our effective feature extraction pipeline, create a scenario where the face and non-face classes are likely well separated in our feature space, playing to SVM's

core strengths in binary classification.



(a) EdgesPCA Polynomial Image 3



(b) EdgesPCA Polynomial Image 4

Figure 42: SVM EdgesPCA Polynomial Results Image 3 & Image 4

The trend from the first two images follows into the third image, with the model maintaining its strong performance in this more challenging group photo, achieving a recall of 1.0 by detecting all faces and an accuracy of 0.889. While the model generates one false positive, resulting in slightly lower precision, the IoU scores remain consistently above 0.60, demonstrating reliable bounding box placement despite the varied face positioning and overlapping subjects in the image.

The model demonstrates impressive performance on the final challenging large group image, successfully detecting 47 faces across a complex scene. With an accuracy of 0.758 and precision of 0.904, the model maintains reliable detection despite the significant increase in complexity. The 10 false negatives primarily correspond to faces that are either partially obscured or posed differently, representing

inherent limitations rather than model failures. Particularly noteworthy is that the model generated only 5 false positives while evaluating 132,924 sliding windows, highlighting its effective discrimination capabilities in a computationally demanding scenario. The IoU scores for successful detection range from 0.7 - 0.93, indicating accurate detection bounding box placement even in crowded conditions where faces are closely positioned.

Given the substantial task of evaluating 132,924 windows, the model's impressive performance and time complexity demonstrate an excellent balance between accuracy and computational efficiency, outperforming many of the other high performing models we investigated, while maintaining reasonable real-world performance.

With this being our optimal model, we decided to comprise the results into a table to showcase it's performance across all 4 images. This is illustrated in Table 9 below:

Table 9: SVM Polynomial Performance Across all Detection Images

Model	TP	FP	FN	Accuracy	Precision	Recall
SVM Poly EdgesPCA	77	6	10	82.80%	92.77%	88.51%

8 Feed Forward Neural Network

Neural networks have gained significant popularity in recent years due to their remarkable ability to tackle complex tasks, often outperforming traditional machine learning models discussed in this report. However, their effectiveness heavily depends on large amounts of training data. Given the constraints of our dataset, which consists of only 124 images, using a neural network may not be the most practical approach. Nonetheless, we explore this method at a high level, primarily for educational purposes and to fill our curiosity.

We opted for a simple single hidden layer feed forward neural network with the following architecture shown in Figure 43:

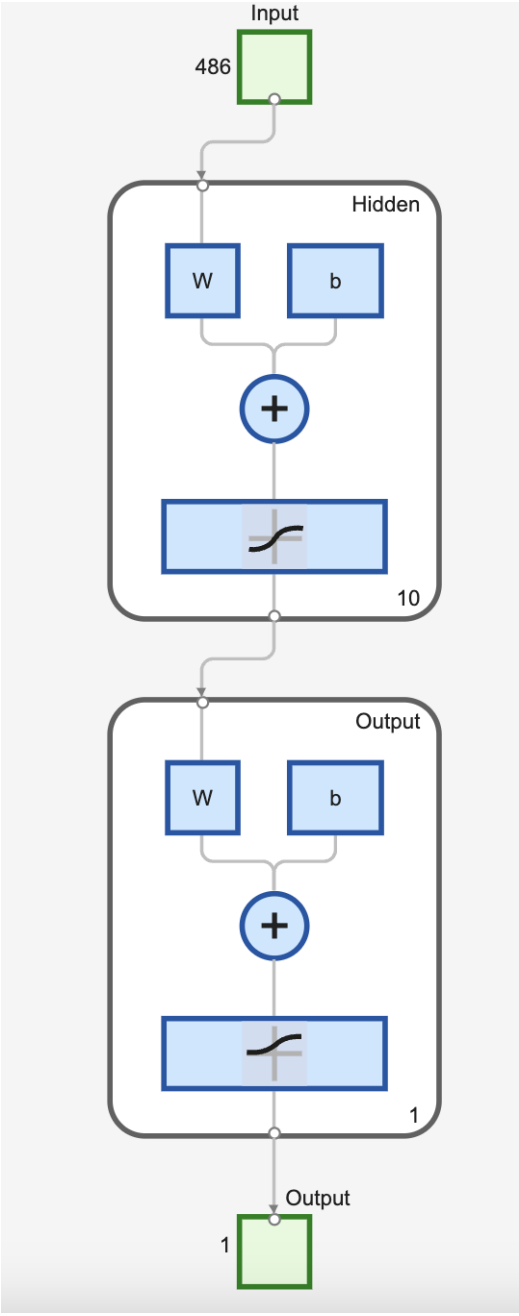


Figure 43: Feed Forward Neural Network Architecture

8.1 Input Layer (486 inputs)

The input layer here receives the input data. In our case, the input features are flattened raw pixel values of face images. Each input neuron represents one pixel, so for images flattened into 486-dimensional vectors, the input layer has 486 neurons.

8.2 Hidden Layer (10 neurons)

The hidden layer introduces non-linearity and allows the model to learn complex patterns from the input data. In complex problems you would generally use many hidden layers (Deep Neural Network), however, in our case we have much too small a dataset for such an architecture and just use one hidden layer to avoid over-fitting the training set.
Weights (W): Each input is connected to every neuron in this hidden layer through weights. This results in a weight matrix of size 10x486.
Bias (b): Each neuron in the hidden layer has a bias value that shifts the weighted sum.
The calculation at each hidden neuron is as follows:

$$\mathbf{Z}_{hidden} = \text{tansig}\left(W_{hidden} \times input + b_{hidden}\right) \tag{3}$$

Where tansig is the hyperbolic tangent function ($tanh$), squashing the values to a range [-1, 1] illustrated in Figure 44. This is the activation function for the hidden layer.

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

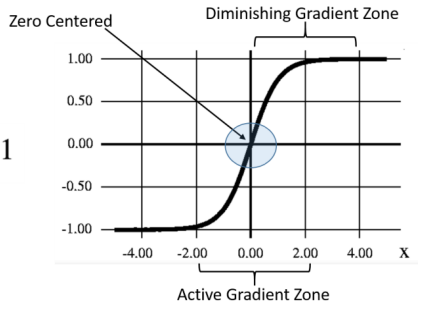


Figure 44: Tansig Function

8.3 Output Layer (1 neuron)

The output layer produces a single output value, which in this case represents the probability of the input being a face in our binary classification problem.
Weights (W): The hidden layer's output is connected to this neuron via another weight matrix.
Bias (b): A bias value for the output neuron shifts the final weighted sum.
The calculation at each output neuron is as follows

$$\text{Output Probability} = \sigma\left(W_{out} \times \mathbf{Z}_{hidden} + b_{out}\right) \tag{4}$$

Where σ is the log-sigmoid function, squashing values to the range [0, 1] illustrated in Figure 45. The activation function for the output layer in our case.

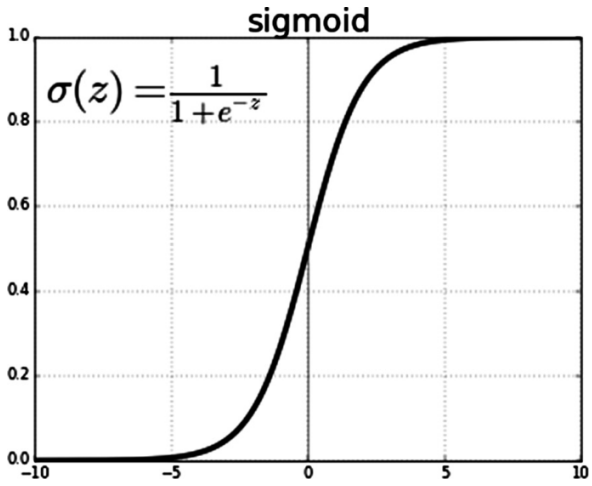


Figure 45: Sigmoid Function

8.4 Training Process

The network uses the cross-entropy loss function to measure how well the predicted probabilities match the true binary labels (face or not face):

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)\right) \tag{5}$$

Where y_i is the true label and $\hat{y}_i$ is the predicted label. We then utilise a Levenberg-Marquardt optimisation to minimise the loss by updating the weights and biases during back propagation. The Levenberg-Marquardt algorithm uses a blend of gradient descent and Gauss-Newton optimisation to minimise the loss.

8.5 Prediction

After the process has been completed, our model can now make probabilistic predictions on test data. Our probability threshold is set at 0.5:

$$\hat{Y}_{\text{pred}} = \begin{cases} 1 & \text{if } \hat{y} \geq 0.5 \\ 0 & \text{if } \hat{y} < 0.5 \end{cases} \tag{6}$$

8.6 Evaluation

After training our model on an augmented version of the training set, we tested it on the hold out set and achieved the following results:

Table 10: Feed Forward Neural Net Evaluation

Model	Accuracy	Precision	Recall	F1-Score	TP	TN	FP	FN
Neural Net	70.00%	0.65	0.87	0.74	13	8	7	2

The neural network performs reasonably well on the test set, though not exceptionally, as anticipated. This can be attributed to its limited depth and the relatively small size of the training dataset, as previously discussed. An area under the curve (AUC) of 0.71 in Figure 46 indicates that the model is notably better than a random classifier, making it worth exploring in the detection phase even if we don't have high hopes. The higher recall compared to precision also suggests that the model is favouring quantity over quality in terms of face predictions, which is not ideal for the detection task. None the less, we will run the neural network against one of our simple detection images and see how it performs compared to our other models.

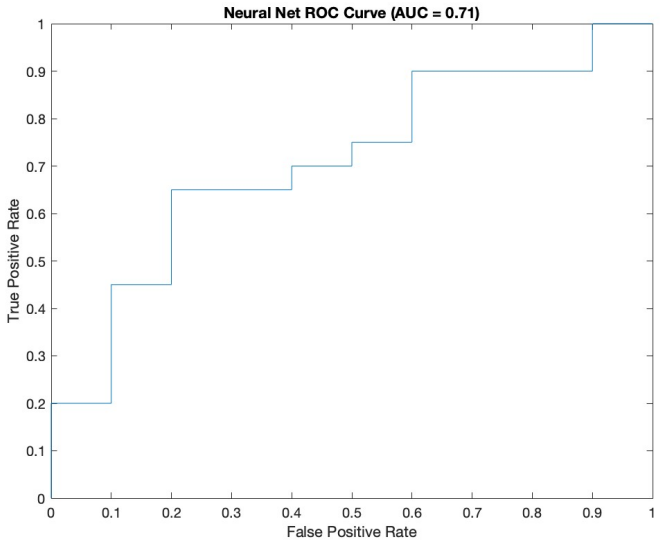


Figure 46: Neural Network ROC Curve

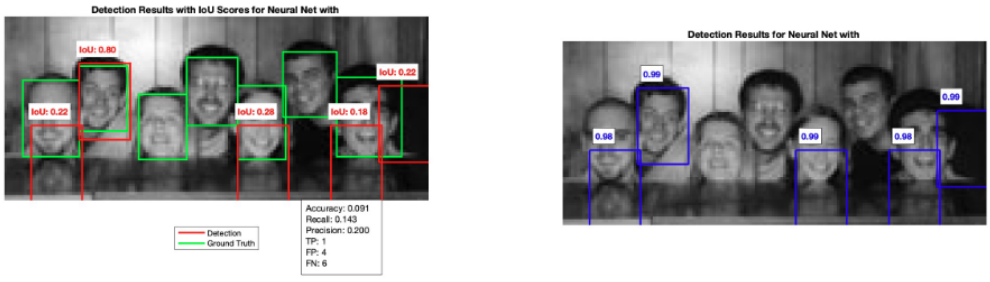


Figure 47: Neural Network Detection Image 1

8.7 Detection

As we can see from Figure 47, the neural network performs quite poorly, only correctly identifying 1 face and misclassifying 4 non-faces as faces while missing the 6 other faces. This is as expected due to the small training set and results we saw in testing of high recall vs low precision. The belief here is that with a much larger and diverse training set a (deep) neural network would perform exceptionally well at trivial detection images such as our detection image set. However, given the constraints on our data-set it makes more sense for models such as SVM to perform better here as they do not rely on having a large training set to perform well.

9 Final Conclusions

The findings from this study are consistent with expectations, as SVM emerged as the best-performing model for face detection. Looking ahead, we aim to further optimise our SVM models by utilising a more diverse dataset and further parameter tuning. We strongly believe the largest contributor to the limitations of each of our models' performance is the training dataset. Given a much larger dataset including higher quality images with varied face angles, illumination levels, distances from the sensor, and more varied examples of non-face images will help enhance the model's ability to perform effectively across a broader range of real-world scenarios.

Additionally, exploring more advanced data augmentation techniques, rather than relying solely on the provided 10:4 face-to-non-face augmentation technique, may lead to more balanced classifiers and mitigate the tendency to over-predict faces; a pattern frequently observed in our results. We briefly explored techniques such as over-sampling non-faces, utilising a 4:4 augmentation split and applying more sophisticated augmentations such as the introduction of artificial noise. However, due to time constraints, we were unable to explore these approaches in depth. This is an area we plan to investigate further in future work.

Furthermore, we aim to explore more effective ensemble methods, such as AdaBoost, which is well-suited for face detection tasks. AdaBoost's dynamic ability to focus on hard-to-classify examples during training helps refine decision boundaries and capture subtle or challenging features. Unlike Random Forests, which treat all samples equally, AdaBoost assigns greater weight to misclassified examples. A notable example of AdaBoost's effectiveness in face detection is the Viola-Jones algorithm [7], which leverages boosting in combination with Haar-like features. We believe implementing a similar approach, incorporating boosting and Haar-cascade features, would yield strong performance in this context.

Another possibility would be to explore incorporating a part-based detector, which would allow the model to identify faces by recognising and combining key facial components across different regions of an image. This approach would increase the model's ability to focus on distinct facial features and not over generalise.

In conclusion, we believe the pipeline we have developed is strong. Expanding and diversifying the dataset, while reducing correlation, would enable more effective cross-validation and fine-tuning, leading to an optimal model after hyper-parameter optimisation for truly unseen data. With these improvements, we are confident that the pipeline can produce a more accurate and resilient model for face detection, capable of performing reliably across a broader range of environments and scenarios.

List of Figures

List of Tables

References

[1] A. Herve, "Principal Component Analysis." https://wires.onlinelibrary.wiley.com/doi/epdf/10.1002/wics.101. Accessed: 2024-10-31.

[2] T. Ahonen, "Face Recognition with Local Binary Patterns." https://link.springer.com/chapter/10.1007/978-3-540-24670-1_36#citeas. Accessed: 2024-11-20.

[3] D. Shanmugam, "When and Why Test-Time Augmentation Works." https://arxiv.org/pdf/2011.11156v1/1000. Accessed: 2024-11-25.

[4] D. Huchaiah, "Grid Search Hyperparameter Optimisation." https://www.tandfonline.com/doi/epdf/10.1080/1206212X.2021.1974663. Accessed: 2024-11-25.

[5] J. Brownlee, "Introduction to Model Selection." https://machinelearningmastery.com/a-gentle-introduction-to-model-selection-for-machine-learning/. Accessed: 2024-10-30.

[6] C.-W. Hsu, "A Practical Guide to Support Vector Classification." https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf. Accessed: 2024-11-28.

[7] ViolaJones, "Rapid Object Detection using a Boosted Cascade of Simple Features." https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=990517. Accessed: 2024-11-30.