Computer Vision Coursework – Project Report

Toby Staines

1. Overview

This report details the development and testing of two functions in Matlab; RecogniseFace and detectNum. RecogniseFace is a face detection and recognition function, which utilises one of six built in classifiers to identify a face, or faces, present in a still image. detectNum is a character recognition function, which can detect a single multi-character number in an image or video, or multiple multi-character numbers in an image.

Both functions were designed for use with a dataset of images and videos of the 2018 Computer Vision class at City, University of London. When tested against this dataset, both functions achieve a high level of accuracy: detectNum scores over 99% and RecogniseFace over 85%, although this is significantly less than its constituent classifiers scored in their development (with the best scoring 97%), and this difference remains largely unexplained.

2. Data Preparation

The original dataset consists of 498 still images and 459 videos of individual class members, split in to 53 folders, plus 36 images and 14 videos of the group. To prepare this data for the training of face classification models, the Matlab script prepareData.m was run.

First, this script copies the folder structure of the original data in a new folder called vidFrames. It then goes through each video in the data, splits it in to its constituent frames and saves each frame in to the appropriate subfolder. It then creates another new folder called facesFolder, copies the sub-folder structure again. Images were loaded using a function called loadImages.m, which borrows from code found online[1] to check and correct the orientation of images. Several of the images were rotated 90 degrees, which caused issues for the development and testing of both main functions.

To detect faces in the images, three different face detectors from Matlab's cascade object detector are used. First, images are converted to 1024x768 pixels, and the default detector, frontal face CART is used. This uses Haar features, and is trained to detect upright, front on faces. If this fails to detect a face, the frontal face LBP detector is used, which is designed to be robust to variation in illumination. If this also fails to detect a face, the profile face detector is used.

In order to speed up detection a minimum size of 100x100 and a maximum of 250x250 pixels is used. Detected faces are converted to 200x200 grayscale and saved to the appropriate subfolder in facesFolder. The same face detection process is then run on the still images from the original dataset and the faces saved in facesFolder. Faces are then split 80/20 in to training and testing datasets.

This approach worked well for the individual still and video images, where only one face per image was expected, and the subject was fairly close to the camera. In the group images there are many subjects in the background, so the minimum filter constraint of 100x100 resulted in many faces being missed. The lower quality of the video frames also meant they were not ideal for extracting faces. To extract a set of faces from the group images all three detectors were run over the data and

all detected faces saved. This resulted in the same face being extracted two or three times from some photos, but the different face detectors do not crop the images identically, so there is still variation in these images, and the repetition acts as a form of data augmentation.

This resulted in a data set consisting of 35,074 labelled faces and 2,762 unlabelled faces. The unlabelled group faces were then spit into training and test sets, and HOG and LBP features extracted. K-means clustering was carried out on these features and the clustered imaged saved in a new file.

Subjective inspection of the clustered images showed that HOG features had resulted in better clusters. These images were then manually assigned to classes using Windows Explorer, and a 54th class 'Unknown' was created for individuals who appear in the group images but not in the individual images. This process was very time consuming (several hours work), but necessary to produce a training set for the 'Unknown' class. Clustering did make the manual sorting easier but clusters were far from perfect. Including photos from the group images in training data for all classes likely also improved performance of all classifiers when tested on other group images, although this was not examined quantitatively.

3. Face Recognition

3.1. Classifier Training

In total, six different classifiers are included in the final delivery, developed using the code in the file extractFeaturesAndTrainModels.m. All six were trained using a set of 21,205 faces, with a validation set of 9,083 and a test set of 7,037.

3.1.1. Convolutional Neural Network (CNN)

The CNN trained is based on the Alexnet model, using transfer learning[2]. The final three layers of the existing Alexnet model were removed and replaced with new classification layers suitable for the 54 classes in the dataset. The Matlab Image Data Augmenter was used to feed images in to the network. Augmentation changed the images from 200x200x1 format to the 227x227x3 format required by Alexnet. It also added random rotation of between -30 and 30 degrees and random translation in X and Y of between -3 and 3 pixels. The validation data was used to monitor for overfitting and the model was trained for 6 epochs.

3.1.2. Feature Types

For the remaining model types, features were first extracted from the images and used to train the models. This should increase the accuracy of the models, by focusing them only on important information in the images and providing invariance to shift, rotation, etc., but it is also necessary to avoid training time and system memory problems. I attempted to provide a baseline by training a Support Vector Machine (SVM) on a vectorised representation of the raw images, in order to demonstrate the improvement gained by using extracted features, but it was not even possible to train the model, as vectorising the images required more than the 8GB of system memory available.

Two feature types were extracted from each of the training, validation and test sets. Histogram of Oriented Gradients (HOG) features were extracted using the extractHOGFeatures function, with nine bins and a cell size of 10x10. This means that the image was broken in to 10x10 pixel cells and the dominant gradient for each cell calculated as belonging to one of nine bins. Within a local block of cells, a histogram of the resulting counts of cells in each bin forms a reduced size representation of the image. Even with this reduction, the default cell size of 8x8 resulted in a training feature set of

over 2GB; too large to save to file from Matlab, even in compressed form. Increasing the cell size resulted in a feature matrix of 1.9GB.

Speeded-Up Robust Features (SURF) were extracted from the training set using the bagOfFeatures function, which extracts features from all images to form a 'dictionary' of all the feature types present. Counts of the occurrence of these features in each image then make up the new representation of the data. Once the bag of features had been created, each of the datasets was encoded and the SURF representations saved to file. SURF relies on the subject of an image taking up the majority of the scene. As the classifiers will be given closely cropped faces of similar scale to the training data, this should make SURF an effective feature extraction method for this task.

3.1.3. Support Vector Machines (SVMs)

Two SVM models were trained; one using HOG features and one using SURF. At first, I attempted to train the HOG SVM using the fitcecoc function with the auto-hyperparameter optimisation option selected. The training and validation sets were merged and a fivefold cross validation was to be used to select the best model. Unfortunately, this optimisation proved very slow; after 24 hours no result had been returned, so the operation was cancelled and a model trained using the default settings. These still produced a very accurate model (94.5%). The SURF SVM was created using the trainImageCategoryClassifier function with default parameter settings.

3.1.4. Random Forests

Random Forest models were trained using both HOG and SURF features. Forests of 50, 100, 150 and 200 trees were trained and then tested on the validation data. Increasing the number of trees showed diminishing returns beyond 100, so a preference for a simpler, faster model was made and the 100-tree model was kept in both cases. Results from this process are shown in Table 1.

Tree Count	SURF Validation Accuracy (%)	HOG Validation Accuracy (%)
50	98.55	97.86
100	98.88	98.13
150	98.95	98.45
200	99.01	98.52

Table 1: Random Forest optimisation results.

3.1.5. Naïve Bayes

Naïve Bayes models were selected as a final option, although were expected to act as a baseline, rather than produce results which would compete well with the other models. The HOG model was trained first, and as expected performed less well than other model types, but was relatively quick to train. Even so, given the previous problems with automated hyperparameter optimisation, and the fact that this was only to be a baseline model, the decision was made to train a single model using the default parameters, rather than attempt to optimise.

Training a model using SURF features resulted in an error, due to some features having zero variance within some classes. Since the HOG Naïve Bayes model had already demonstrated significantly worse accuracy than other model types, it was decided not to continue with the SURF Naïve Bayes option.

3.1.6. Mood Detection

The planned approach to detecting mood in images was to copy the dataset (or a portion of it) and manually reclassify images into {happy, sad, surprised, angry}. This new dataset would then be used to train another CNN, based on Alexnet, to be run on detected faces following the ID classifier.

However, in the absence of labelled data, this would have been a very time-consuming task, and was deemed infeasible.

3.2. Results

Table 2 shows the test results of each model trained. As expected, the CNN model outperformed all others, although all model types, apart from Naïve Bayes, demonstrated accuracy of over 90%. The HOG feature vectors extracted were significantly larger than the encoded SURF datasets, meaning that training on HOG features took significantly longer than using SURF.

Model	Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
CNN	100	99.28	97.42
HOG Random Forest	100	98.13	95.05
HOG Naïve Bayes	92.32	91.65	84.19
HOG SVM	100	98.58	94.74
SURF Random Forest	100	98.88	95.77
SURF SVM	96.92	96.61	92.90

Table 2: Classification model training results.

Figure 1 shows some examples of faces incorrectly classified by the CNN model, with an example of the true class in the first column, a misclassified image from that class in the middle, and an example from the predicted class on the right. In the first row we see that the input image was highly blurred (extracted from video), and it is unsurprising that the classification was problematic.

In the second row, one could subjectively say that the two people in question look fairly similar (both male, similar hair, both wearing glasses). Additionally, class 65 has far less data than most classes (only 16 training images, after the addition of faces from the group photos), and had only three images in the test set, all of which were misclassified. This highlights the problems with inconsistent data volumes across classes, and low volumes for any class. This could be addressed with further data augmentation, but in situations where some classes have 20 or 50 times as many examples as others this is an imperfect solution, as there is only so far you can augment such a small dataset – you are essentially still sampling the same limited distribution. This is reflected in the confusion matrix of the results (ConfusionMatrix.xlsx), where some classes, especially those with a smaller dataset, perform significantly below the overall level of the classifier.

The bottom row shows a misclassification that is hard to explain in human vision terms. The example from the true dataset appears very similar to the misclassified image, and was itself part of the test set and correctly classified. The confusion matrix does show that the classifier had some problems with class 05, achieving only 85% accuracy despite the class containing almost 600 training images.

Figure 1: Examples of faces incorrectly classified by the CNN model.

3.3. RecogniseFace

3.3.1. Requirements

The RecogniseFace function should:

Requirement Description	Must/Nice to Have	Delivered
Accept as input a single UINT8 object	Must	Y
Accept an argument classifierName, with a value in {'CNN','SVM','RandomForest','NaiveBayes'}	Must	Y
Accept an optional argument featureType, with a value in {'NIL','HOG','SURF'}	Must	Y
Detect all faces in input image	Must	Y
For each face detected, return the coordinates of the centre of the face	Must	Y
For each face detected, return the ID of the individual	Must	Y
For each face detected, return the mood of that person in {happy = 0; sad = 1; surprised = 2; angry = 3}	Nice to Have	N

Return and updated version of the input, annotated with a bounding box around each detected face, labelled with the predicted ID of the face.	Nice to Have	Y
Optionally display the annotated version of the input, depending on the value of displayAnnotatedImage	Nice to Have	Y

Table 3: Requirements for RecogniseFace

3.3.2. Instructions for use

RecogniseFace should be called as follows:

[P, annotatedImage] = RecogniseFace(I, classifierName, featureType, displayAnnotatedImage);

I and classifierName are mandatory inputs; featureType is optional and defaults to 'NIL', displayAnnotatedImage is optional and defaults to true.

The table below lists all valid combinations of classifierName and featureType. Any other combination will result in an error.

classifierName	featureType
CNN	
CNN	NIL
SVM	HOG
SVM	SURF
RandomForest	HOG
RandomForest	SURF
NaiveBayes	HOG

Table 4: Valid argument combination for RecogniseFace.

3.3.3. Description

RecogniseFace first checks the number of optional arguments, and the values of classifierName and featureType passed. If any these are invalid an error will be produced. It then scans the image using a sequence of face detectors.

During data preparation, all individual images passed to the face detector were converted to a standard size, but RecogniseFace cannot do this, as it does not know whether the image is of a group or of an individual. Compressing group photos to a standard size risks making background faces harder to detect and/or classify. Removing the bounds on the size of face which the detector was searching for increased the detection time to over 10 seconds per image, which was deemed too slow. Instead, wider bounds are used than in data preparation to account for the greater variability in the group images; a minimum of 60x60 and a maximum of 325x325 is used. However, this caused another problem, due to the different resolution in the sample images; even with an increased window size of 325, many faces in the individual images began to be missed, as they were too big. In the final function, the first face detector (Full Frontal CART) is used on the image as provided. If no face is detected in the image it can be assumed to be of an individual, so is resized, and the three face detectors in sequence are used, using the original face size parameters. This succeeded in reducing the average time for the face detection process to less than two seconds per image, whilst maintaining a high success rate.

A more sophisticated approach to handling the results of multiple face detectors would be useful for use on group images, where currently only one detector is used, which is relatively weak when faced with occluded or profiled faces. A better way to deal with variation in input image resolution without impacting performance would also improve the function.

Detected faces are converted to the standard format of 200x200x1 and stored in a gallery. The empty matrix P is then created, with one row per detected face. Face centres are calculated using the bounding box values returned by the face detector.

The function then loads the selected classifier, based on the classifierName and featureType specified in the call. If an invalid combination has been passed an error will be returned.

The faces in the gallery are then passed to the selected classifier and the results stored as numeric values in the first column of P. Depending on the classifier used and the way it was trained, some processing is required to translate the model output into a scalar value.

Finally, bounding boxes are inserted in to the original image around each detected face, labelled with the ID returned for that face by the classifier. This annotated image can be stored by using the

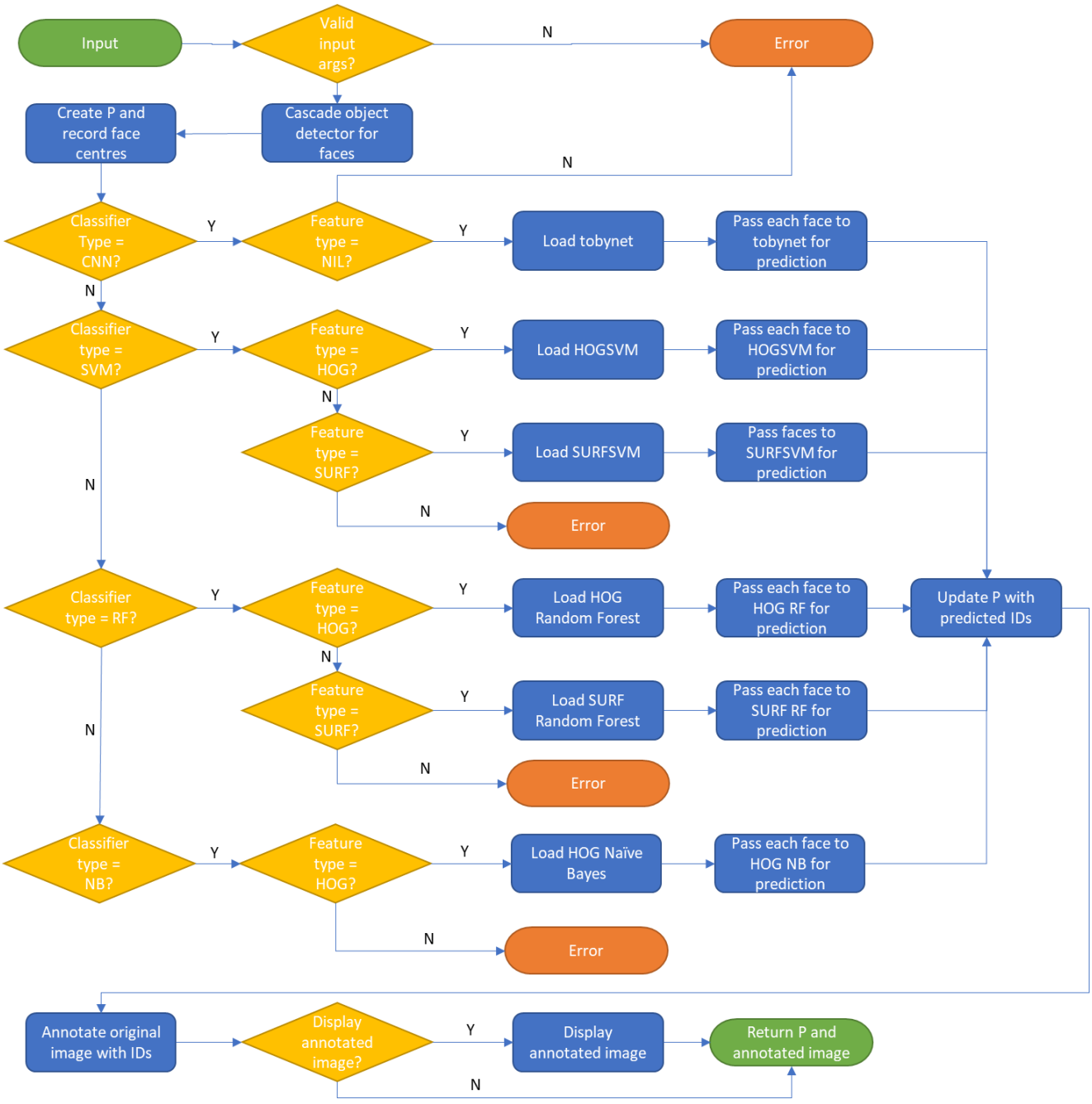

Figure 2: Block diagram describing the RecogniseFace function.

optional output annotatedImage, and will be automatically displayed when the optional argument displayAnnotatedImage is set to true (which it is, by default). This was not required in the specification, but was useful during development and was retained as a nice additional feature.

3.3.4. Results and Discussion

The final version of RecogniseFace was tested on the 498 original individual photos for each classifier and the results are given in Table 5. The expectation was that RecogniseFace, with the CNN classifier selected, would perform as well or better than in the classifier's training, since this test involved none of the more tricky group photo images, and, more importantly, many of the images in this data would have been the source of faces which were in the network's original training set. However, initial experiments showed accuracy for the overall function reduced to 81.4%. Inspection of the code showed only one difference in the pre-processing steps between developing the CNN and running the function: during training, faces were extracted and then saved to file as a jpg, before being re-loaded later for training. RecogniseFace did not do this; extracted faces were passed straight in to the classifier (following size standardisation and conversion to grayscale). Addition of this save and re-load step did increase accuracy by 4% to 85.7%, but this is still some way below the 97.4% achieved in the model's initial test. In fact, all models showed a reduction in accuracy from their original testing, but to varying degrees, meaning that the CNN model was no longer the most accurate, with both SURF models now showing better results. The save and reload step has been commented out in the final submission to supress production of many jpg files when testing, but can easily be reinstated if desired.

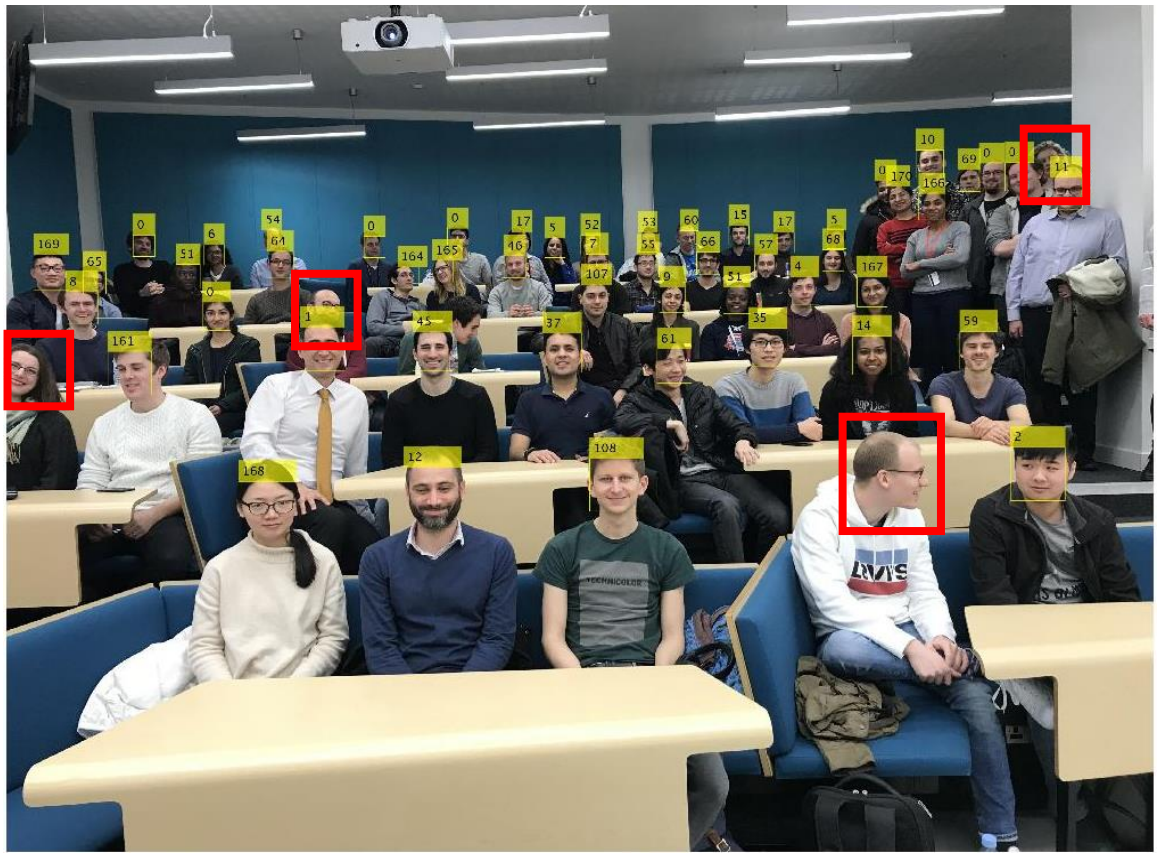

Figure 3: Face detection and classification using a CNN. All but one of the detected faces are correctly classified, but four faces have been missed by the detector (red boxes); two due to occlusion, one due to being in profile, and one most likely due to being tilted, and possibly because it is slightly out of frame.

Model	Accuracy (%)	Predict Time(/Image)
CNN	85.74	1.81
HOG Random Forest	79.32	1.84
HOG Naïve Bayes	74.70	10.45
HOG SVM*	80.98	142.99
SURF Random Forest	86.35	1.83
SURF SVM	89.16	0.80

Table 5:Testing results for RecogniseFace on 498 still photos of individuals.

Since the majority of training and testing data was extracted from video frames, rather than still images, it is possible that there is some difference between these two types of image, which is not obvious to the human eye, but has affected the model's training, and it is overfit to faces from video.

Table 5 also shows the average time taken for RecogniseFace to process a single person image using each of the classifiers. The SURF SVM is comfortably the fastest, with the CNN, and Random Forest models about twice as fast again, but still with an acceptable time of under two seconds. The Naïve Bayes model is five times slower again and the HOG SVM takes a massive 142.99 seconds per image. It is likely that using a smaller HOG vector would improve this performance, so further optimisation work is required here, but the largest bottleneck is loading the model, which is almost 200MB in size (even in compact form). This is a drawback of the fitcecoc function, as other model types using the same HOG features do not have this problem. Reengineering the function to accept multiple inputs at once, rather than being called repeatedly for each image, would greatly improve the performance on large data sets, although the issue would remain for handling single images.

Figure 4: Face detection and classification using a CNN. All faces which are fully in frame have been correctly detected, but two have been misclassified (red boxes).

The lack of a labelled ground truth for the group images mean that no quantitative test has been conducted on this data. Qualitatively, the results are of a slightly lower level than the individual images. Examples of annotated group images are provided in Figure 3 and Figure 4. The majority of faces have been detected successfully, but some are missed (Figure 3) and some have been misclassified (Figure 4).

RecogniseFace was also tested on several images which had no people in them. On landscapes, nothing was returned, but on photos including closeups of signs, some of the characters were returned as faces. There is no confidence threshold function available for the cascade object detector, so once a face is detected, even if it is not a face, it is passed to the selected classifier. Once this happens, the classifier has to make a decision about which of the 54 classes to put it in, and currently the function has no good way to handle this. An improved approach might be to use the softmax scores (or equivalent probabilities from the other classifiers) to set a minimum confidence threshold for classification, and return a second category of unknown if it is not met, rather than the best guess ID, which may be almost a random choice. This approach would also help to handle 'unknown unknown' people – those who were not in the individual or group data sets but are in other photos. There was at least one such person present in the final lecture of term when additional test photos were taken.

4. Optical Character Recognition (OCR)

4.1. Requirements

The detectNum function should:

Requirement Description	Must/Nice to Have	Delivered
Accept as input the filename/path of a .jpg file	Must	Y
Accept as input the filename/path of a .mov file	Must	Y
Accept as input a UINT8 image object	Nice to Have	Y
Accept as input a videoReader object	Nice to Have	Y
Detect a single number (possibly of multiple digits) in an input image	Must	Y
Detect a single number (possibly of multiple digits) in an input video	Must	Y
Detect multiple numbers (possibly of multiple digits) in an input image	Nice to Have	Y
Detect multiple numbers (possibly of multiple digits) in an input video	Nice to Have	N
Return all numbers detected from a valid input	Must	Y

Table 6: Requirements for detectNum.

4.1.1. Instructions for use

detectNum should be called as follows:

num_vec = detectNum(inputMedia, singleNumOnly);

inputMedia can be a uint8 image, a VideoReader object or a file path.

The singleNumOnly argument is included mainly to supress an attempt to detect multiple numbers in videos, which requires further development work. singleNumOnly is optional and defaults to false. When set to true, detectNum will return the number detected with the highest confidence in

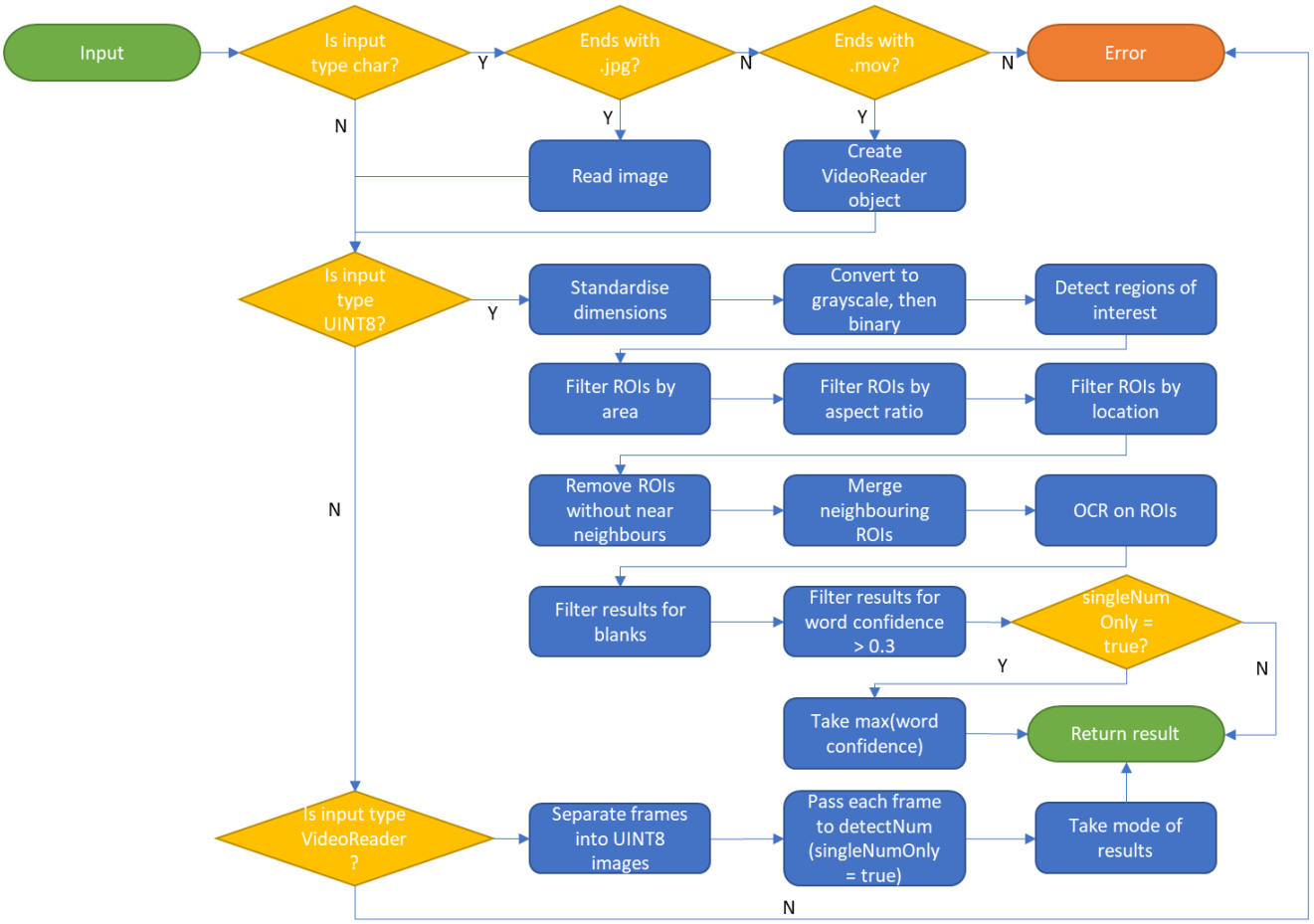

Figure 5: Block diagram describing the detectNum function

images. When set to false (or not entered by the user), detectNum will return a vector of all numbers detected in an image. The function will always return only a single number from a video.

4.2. Description and Results

The detectNum function uses Matlab's built in OCR function as a foundation. Passing raw images to OCR resulted in 0% successful number detection, but with considerable pre-processing of images it was sufficient to achieve over 99% accuracy on the provided data-set, for both image and video input.

Much of this pre-processing is highly specific to the provided data-set, so it should be noted that the detectNum function is not expected to generalise well to scenarios where numbers are in different settings or formats, or not in the centre of the image. A more robust approach may be to use a neural network trained on a variety of number types and use a varying size, moving window on input images as input to this network. However, for the data provided the approach used is more than sufficient.

Several sections of the code for this pre-processing are adapted from pages on the Matlab website[2][3][4][5].

detectNum first checks whether the object passed to it is a character array, and if it is, whether the array ends with '.jpg', in which case it replaces the character array with the loaded image using imread, or '.mov', in which case it replaces character array with a VideoReader object.

Figure 6: Filtering of ROIs from an image. A) ROIs detected. A -> B) ROIs filtered by area. B -> C) ROIs filtered by aspect ratio. C -> D) ROIs filtered by proximity to image edge (see small ROI on right edge of C). D -> E) Neighbouring ROIs expanded and merged. Remaining ROIs are then passed to OCR.

It then checks the object type again. If it is a UNIT8 image it proceeds to the main body of the function. The first step in pre-processing is to standardise the size of the input image to 1024 x 768, and then convert it to grayscale. During development the addition of blurring at this stage was experimented with, but was found to worsen results so was removed.

Binarization (conversion to a binary image) takes place within the OCR function, but I found that binarizing the image at this stage, and then taking the compliment of the result, greatly improved the detection of areas of interest. The binary image is passed to Matlab's blob analyser for detection of regions of interest (ROIs). This was generally very good at picking up the numbers, but also detected a large variety of other ROIs, many of which would result in spurious numbers being returned if they were passed to OCR. Because of this, further pre-processing was required to ensure that only the regions which actually contained the desired numbers are passed to OCR. These steps are illustrated in Figure 6.

The first step in this filtering of detected regions was to apply an area constraint. This successfully removes all of the very small and very large regions, but the constraint needs to be quite broad in order to account for variation in how close the subject of the image is standing to the camera. The final area constraint used (in pixels) is 30 < area < 1100. A further constraint on aspect ratio is then applied to remove very tall and skinny or wide and short ROIs, as these are unlikely to contain numbers. Aspect ratio is calculated as width / height and a constraint of 0.25 < aspect ratio < 1 is used.

Since the target number is always in roughly the same area of the image, any ROIs within 200 pixels of the sides of the image, or within 350 pixels of the top or 100 pixels of the bottom of the image are also removed. In particular this tended to remove subject's heads, where OCR often incorrectly detected ears as 0 or 9.

The final step in pre-processing is to merge overlapping, or very close, ROIs. Since all of the target numbers consist of two characters (with a leading 0 for numbers less than 10), it is reasonable to assume that we are only concerned with regions which have close neighbours; a region which is on its own is not of interest. This section of code has been separated into a function named mergeOverlappingRois. Much of the code in this function is from [4]. This function slightly expands all ROIs and then calculates the overlap ratio between a ROI and all other remaining ROIs. If the ROI has an overlap ratio of 0 with all other ROIs then it is removed. The remaining ROIs at the end of this process are then returned to the main detectNum function.

The final set of ROIs is then passed to the OCR function, with the name-value-pair arguments 'CharacterSet', '0123456789' and 'TextLayout', 'Word'.

A final set of checks on the result returned by OCR remove any results where no number was detected and words where the confidence is lower than 0.3. All of the above results in a single number, or vector of numbers, which is then returned. If no number has been detected the function returns NaN.

When tested on the 498 still images in the dataset provided, detectNum accurately detected the correct number in 99.6% of images. It also incorrectly identified an additional number in 2 photos (0.004%). It was then tested on five pairs of photos which had been joined together to test the ability to detect multiple numbers and achieved 100% accuracy, although clearly this is a very small data set and further testing here would be beneficial.

In the case where a VideoReader object is detected rather than a UINT8 image, the function extracts the individual frames from the video and passes them all through the main function. Since many of the individual video frames are highly blurred the accurate detection rate is much lower than for still images. detectNum collates the results from all frames in a column vector and returns the mode value (not including NaN). This summarisation results in a high level of accuracy at the video level; when tested on the 459 videos provided, detectNum correctly identified the number in 99.3% of cases.

Video frames are always passed to detectNum with the singleNumOnly argument set to true. The video section of the function expects each frame to return a single number, which is then put in to a column vector. This could easily be extended to a matrix, to handle multiple numbers being returned, but this assumes that the same number is always detected in the same order, so as to be returned to the same column of the matrix. As such, extending multiple number detection to videos is non-trivial and requires further work.

5. Conclusion and Further Work

This report outlines an approach to the development of two complementary computer vision systems, for character recognition, and face detection and identification, and highlights some of the challenges of such an approach. The expectation was that a CNN model would prove to be the most adept at classifying faces, and while this was the case in initial testing, the final function did not reflect this, and further work is needed to understand the causes of such a change. Nevertheless,

while not performing to a commercial level, both functions perform their designed tasks well, and make good 'proof of concepts'.

In addition to improvements discussed throughout the report, there are a number of areas in which this work could be extended. Completion of the emotion detection element and extension of multiple number capture to video would complete the full specified functionality.

All of the classifiers used could potentially be optimised further and experimentation with other feature extraction methods, such as Gabor filters, could provide interesting extension. I would also like to extend both functions to accept multiple inputs, in the form of image data stores or cell arrays of images, rather than requiring repeated calls in a for loop, and it is likely that this would bring performance improvements in terms of execution time. detectNum could also be improved by making it robust to a wider variety of characters, both in their form and in their location in the image, which is currently very specific.

Extension of RecogniseFace to work on video input, and of both functions to work on a live video stream would be an interesting area to explore and provide useful additional functionality.

6. References

[1] W. Robertson, "How to load a jpg properly? - MATLAB Answers - MATLAB Central," 2015. [Online]. Available: https://uk.mathworks.com/matlabcentral/answers/260607-how-to-load-a-jpg-properly. [Accessed: 16-Apr-2018].

[2] S. Jalali, "Biologically Inspired Image Classification and Deep Learning." p. 78, 2018.

[3] "Recognize Text Using Optical Character Recognition (OCR) - MATLAB & Simulink - MathWorks United Kingdom." [Online]. Available: https://uk.mathworks.com/help/vision/examples/recognize-text-using-optical-character-recognition-ocr.html. [Accessed: 16-Apr-2018].

[4] "Automatically Detect and Recognize Text in Natural Images - MATLAB & Simulink - MathWorks United Kingdom." [Online]. Available: https://uk.mathworks.com/help/vision/examples/automatically-detect-and-recognize-text-in-natural-images.html. [Accessed: 16-Apr-2018].

[5] L. Shure, "Nice Way to Set Function Defaults » Loren on the Art of MATLAB," 2009. [Online]. Available: https://blogs.mathworks.com/loren/2009/05/05/nice-way-to-set-function-defaults/. [Accessed: 16-Apr-2018].