INM460: Computer Vision Coursework

James Hooper *James.Hooper@city.ac.uk*

April 22, 2018

Abstract

In this project a function called RecogniseFace has been created to classify faces from images. An Error Correcting Output Code Classifier (ECOC) with SURF features was the best classifier when group photos were taken into consideration. Another function called detectNum was created to detect numbers on a piece of card held by people in an image/video. It is very accurate as the location of the number is consistently on a white background, thought there are ways it could be adapted for different scenarios.

1 Face Recognition

N.B. The code used to extract features was adapted from Tutorial 5 and the code used to train the CNN using Alexnet was taken from the Matlab website [1].

1.1 Overview

In this section an overview of the RecogniseFace function is provided. Figure 1 is a block diagram showing the main the algorithmic components that make up RecogniseFace. These will now be covered in slightly more detail.

The initial dataset used for this project was a collection of images for 53 distinct people plus a collection of group images. All images apart from the group images are of one person holding a number. This is the number that will classify them. Only an image of the face is needed to train the classifiers so *Vision.CascadeObjectDetector* was used to extract faces from all of the images and save them into a Face Gallery as seen in Figure 1.

There were as few as three images for some of the labels. In order to train an accurate classifier many more images were needed. To solve this problem, first, eight new images were created with augmented brightness from $I - 100$ to $I + 100$ for each of the original images in the Face Gallery, then for each of the newly augmented images have a gaussian filter applied using imgaussfilt(I, k) for $k = [2, 4, 6, 8, 10]$. In total this created 40 new augmented images for each original image in the Face Gallery and, in theory, this should provide enough variety so that our classifiers do not overfit on the training images.

An Error Correcting Output Code (ECOC) Classifier, Random Forest (RF) Classifier and a Convolutional Neural Network (CNN) Classifier were used as the classifiers for RecogniseFace. ECOC was chosen over a Support Vector Machine Classifier as it performs better in multiclass classification problems. The CNN was trained using Alexnet with custom layers for this problem. Since there is a significant increase in computational time compared to an insignificant increase in accuracy for each classifier, especially the CNN, 40 images were randomly chosen for each label and this was split, using the Holdout method, into 32 training images and 8 test images for each label.

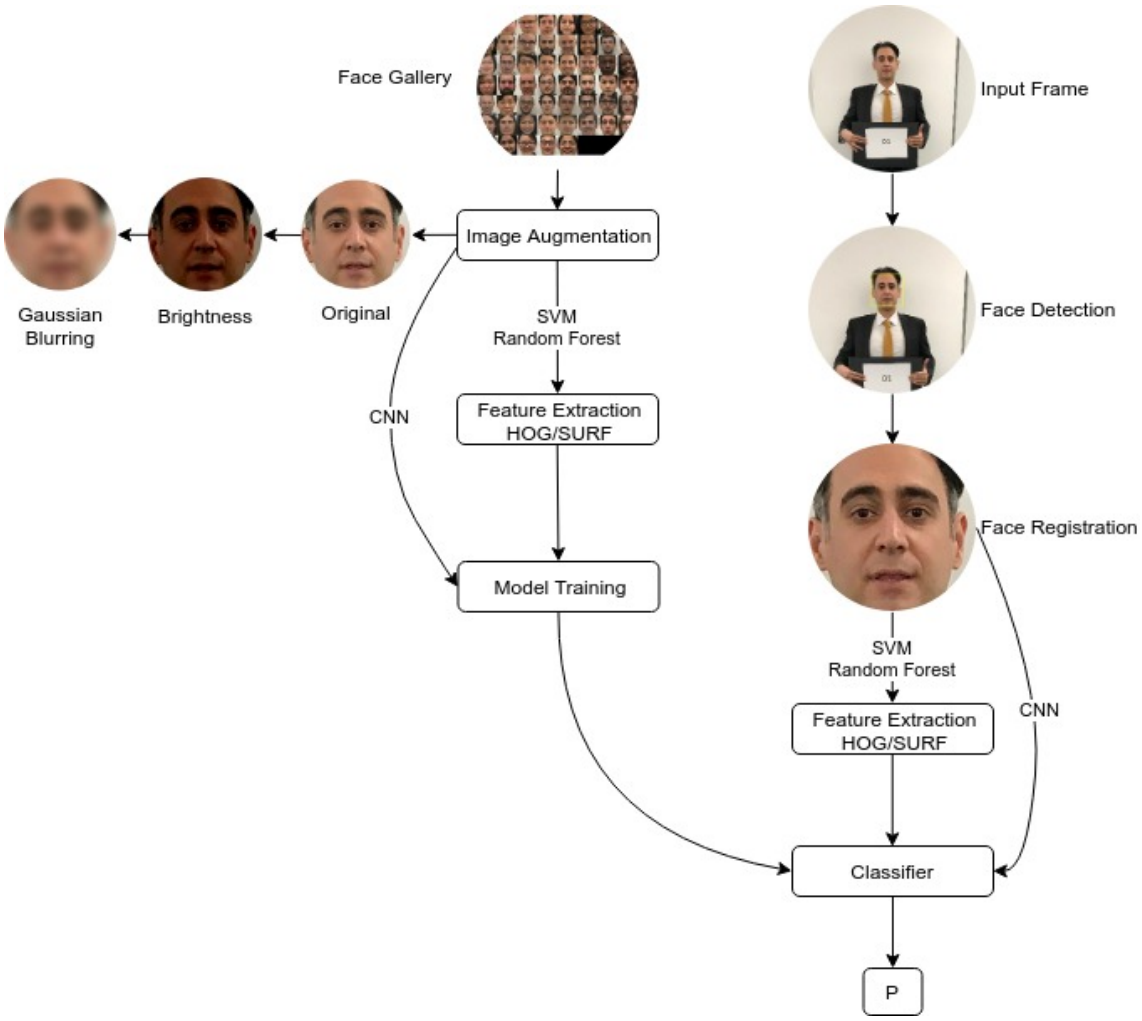


Figure 1: Block diagram for RecogniseFace algorithm

Speeded-up Robust Features (SURF) and Histogram of Oriented Gradients (HOG) were extracted from the set of training images. These features were then used to train two ECOC Classifiers and two RF Classifiers, one fore each feature. Since the CNN does not use features for the input it was directly trained on the training images. Using the testing images, a confusion matrix and accuracy percentage was calculated for each combination of classifier and feature. A more in depth evaluation of the classifiers is in the following section 1.2.

Up until this point everything mentioned is to train the classifiers to be used in the RecogniseFace function, which you can see visually on the left half of Figure 1. The implementation of RecogniseFace is on the right half of Figure 1. RecogniseFace will take an image, detect faces in that image and crop them out so they are of the same size that is used to train the classifiers, so 200 by 200 for ECOC and RF and 227 by 227 for the CNN. SURF (after image has been turned to grayscale) and HOG features are then extracted to be used in the ECOC and RF classifiers, whereas the images are used directly in the CNN. After each image has been classified, the prediction is placed in the first column of P followed by the x-axis value for the center in the second column and the y-axis value for the center places in the third column.

1.2 Requirements

In Table 1, below, the different possible ways of calling P = RecogniseFace(I, featureType, classifierName) are shown, where featureType and classifierName must be strings as show in the table. The test accuracy of these combinations is also shown in the Table.

featureType classifierName	'HOG' 'ECOC'	'SURF' 'ECOC'	'HOG' 'RF'	'SURF' 'RF'	'' 'CNN'
Test Accuracy	99.07%	97.45%	98.38%	97.22%	97.69%

Table 1: Ways to call featureType and classifierName in RecogniseFace with Test Accuracy for each method.

As an example, to call the function RecogniseFace using SURF features and a RF Classifier would be P = RecogniseFace(I, 'SURF', 'RF') and to call the CNN Classifier would be P = RecogniseFace(I, '', 'CNN'). In order to use the RecogniseFace function, a number of other items need to be in the same folder. These are:

- SURF_ECOC_Classifier.mat
- SURF_RF_Classifier.mat
- HOG_ECOC_Classifier.mat
- HOG_RF_Classifier.mat
- CNN.mat
- bag.mat
- personIndex.mat

The best performing classifier going only by test accuracy is the ECOC Classifier using HOG features. However in practice this is not the case. When the input is a group image

and the faces within that image are smaller it is much more accurate to use the ECOC classifier with SURF features. This is most likely because the SURF features are invariant and will have taken into account a change in angle of the persons face better than HOG features. Therefore in Figure 2 there is confusion matrix showing the results for that classifier. It can be seen that there is a column labeled 0, which is the label of all people who were in group photos but did not have individual photos. Most labels had 8 out of 8 correct classifications, a small number of labels had one incorrect classification and only one label, 169, had two incorrect classifications.

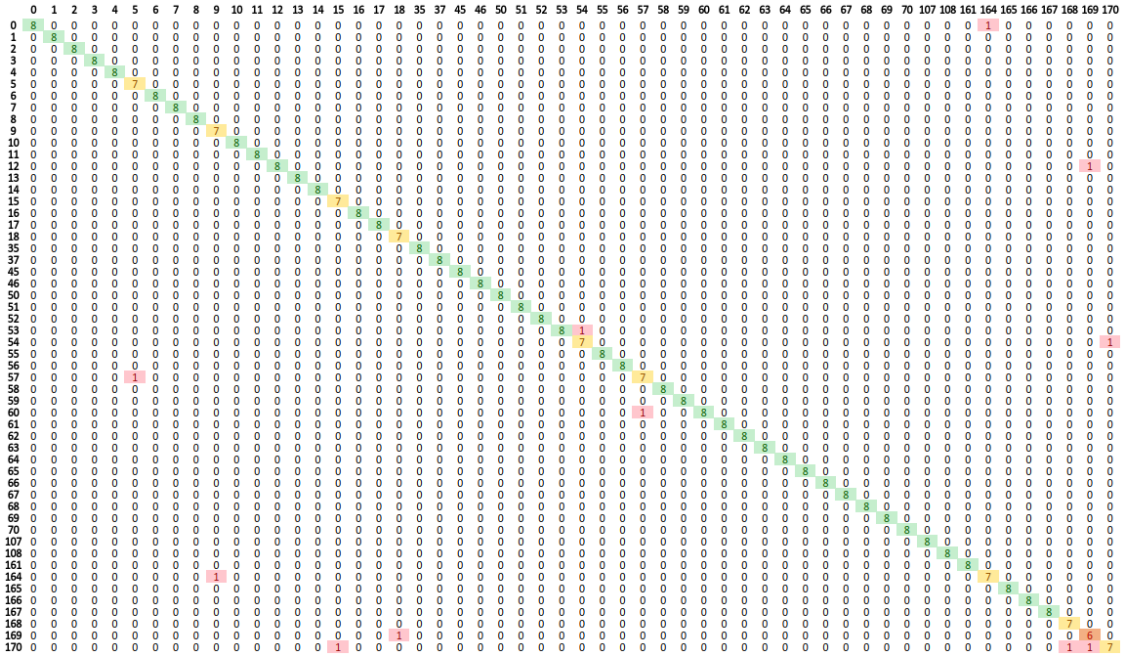


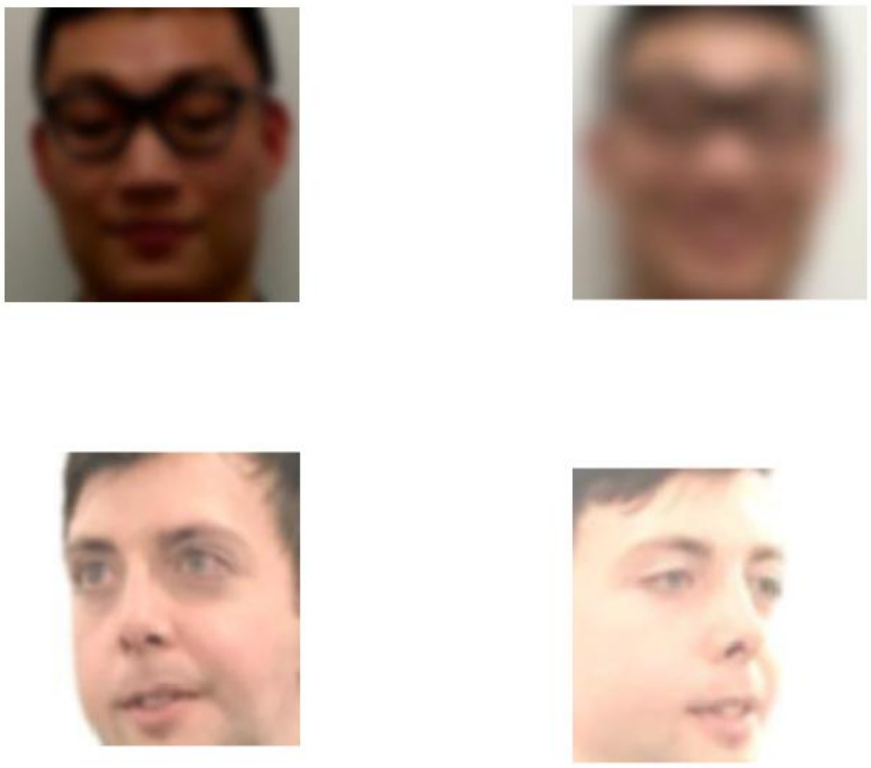
Figure 2: Confusion Matrix for the ECOC Classifier using SURF features (8 test images per label).

In Figure 3, a couple of correct and incorrect classifications are shown. The general trend is that the only incorrect classifications are caused by overly blurred faces or the faces are turned at too much of an angle, both of which may not have been picked up in training as only a small subset of all the total augmented images were used.

1.3 Approaches & Reflection

The final approach, which was used, proved to be very effective in most scenarios. One issue that sometimes occurred was when the input image was of a group and the image was taken from an angle such that the individual faces extracted were of the side of the face. In that case the ECOC classifier with SURF features was not particularly accurate. Another case when the classifier was inaccurate was for faces at the back of a group photo that, once extracted, were very distorted. In this case the ECOC Classifier with SURF Features was not accurate. A final issue was false positive face detection using the *vision.CascadeObjectDetector*, as it sometimes detects areas that are not faces and then they are fed into the classifier. The ECOC Classifier with SURF Features is, however, extremely accurate for images with either one person, or a smaller number of people than is usually in the group photos.

Some potential ideas for future work to improve the accuracy of the RecogniseFace function are as follows:



(a) Correctly classified

(b) Incorrectly classified

Figure 3: Classifications

1. Adding another label called 'NotFaces' and including any false positives that come from the initial image extraction to make the face gallery. In theory, any new false positives will then be classified as 'NotFaces' and could be ignored.
2. Try and find a way to mimic the poor resolution that was found for people at the back of a group image. Unfortunately, though the attempt to use Gaussian Filtering and brightness adjusting worked for some faces closer to the front of the group image, it was not good enough for images at the back.
3. Due to a huge computational time increase, the training set was limited to 32 images per label. If more images were used then more variation in the training set would have been present and may have generalized better to new images.
4. Extracting faces straight from the group image and using these to train the classifiers could also be a good way of classifying faces with poor resolution that are at the back of a group image.

2 OCR

2.1 Overview

In this section an overview of the detectNum function is provided. Figure 1 is a block diagram showing the main the algorithmic components that make up detectNum. These will now be covered in slightly more detail. It is important to note that large parts of the code for this algorithm were taken directly from Matlab [2] and then adapted and added to.

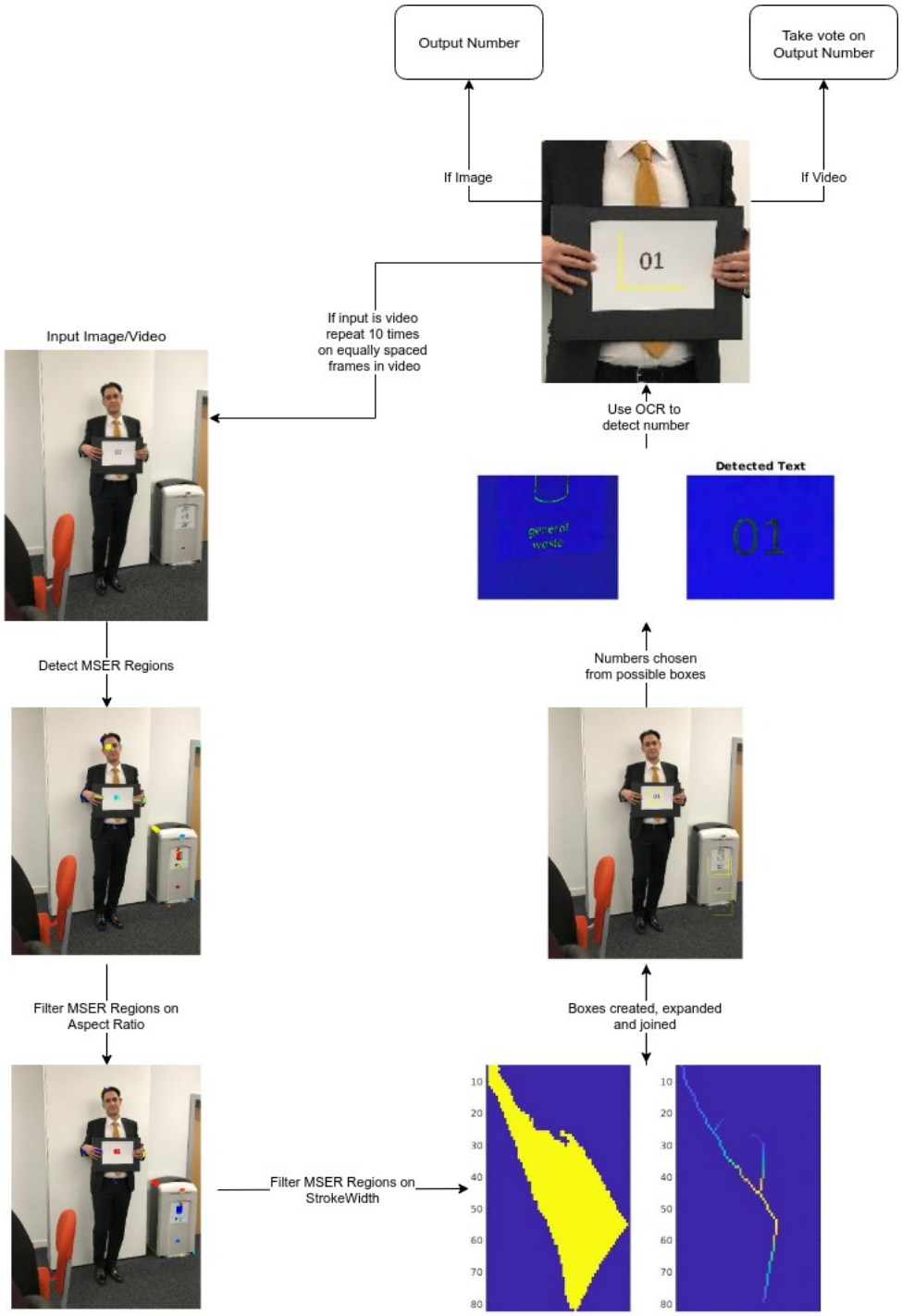


Figure 4: Block diagram for detectNum algorithm

It is easier to start with the process of taking an image in detectNum as videos are just an expansion of the same method. Initially, the function reads the image and detects MSER Regions (a type of blob analysis). These blobs are then filtered by aspect ratio as numbers are more likely to be of a certain aspect ratio with the specific condition being that $height > 3 * width$. The blobs are then filtered on Stroke Width as a number will have a certain width in the line and anything, to a 15% tolerance, which is outside of this width is filtered out. Any blobs that are left have a bounding box drawn around them. These boxes are expanded slightly and every overlapping boxes are joined so that sequences of numbers are in the same bounding boxes. Since that the numbers are printed on a white card, it is simple to filter the leftover boxes to find ones that contain numbers with a white background by converting to HSV and picking the image with the highest percentage of intensity above 0.9. The in built OCR function then reads the chosen square to output a final number.

To detect numbers in a video the process is almost identical. Ten equally spaced frames throughout the video are extracted. The process to detect numbers from one image is then used on each frame, giving 10 detected numbers. The number that is detected most often is used as an output.

Nothing apart from the function detectNum.m needs to be in the directory to call the function. An example of the function being called is detectNum('IMG_0660.jpg').

2.2 Approaches

The final approach, which was used, proved to be very effective in most scenarios for both images and videos. The only issue that sometimes occurred was when a frame from an image was blurry and in that case it was less likely that the number was extracted successfully. This was usually mitigated by the fact that 10 frames were being used A number of alternative approaches were attempted before reaching the final solution:

1. Changing the image to binary in an attempt to identify where the piece of paper was located in the image led to poor results. This was due to the background of most of the images being lighter in colour and therefore too many regions of interest were found and it was difficult to filter and find the ones with characters in them.

2. The next attempt was to train a Cascade Classifier that could extract the piece of paper each person is holding to make it easier for the OCR function. This proved unsuccessful because the classifier picked up too many regions from the image and made it difficult to extract the number.

2.3 Reflection

The strength of the detectNum function heavily relies on the fact that the surroundings of the number are known i.e. it is known that it is printed on a white piece of paper. Once a set of possible regions that could be numbers have been found, this makes it significantly easier to find the region that is most likely to be a number on the card. Therefore using this function as it is to detect numbers in any other scenario would not work.

However the function could easily be adapted for other scenarios where the surrounding of the number was known and also it could be adapted to find any number in the image (independent of the surroundings). The piece of code that would need to be changed is in Figure 5. Essentially, after a number of text boxes have been filtered out and there are only good candidates for text/numbers left, this code chooses the one most likely to be the white background with a number on it. If there were more than one number being held on a white background the code could easily be adapted to choose the top 2,3,4,... etc text

boxes, filtered on the percentage of pixel intensity above 0.9. This could also be adapted to filter on different conditions if the number was known to be on a different background, for example on a darker background the pixel intensity could be filtered below 0.1.

```
for i = 1:size(textBBoxes,1) % run through each box that is left
    subI = imcrop(I,textBBoxes(i,:));
    %subplot(2,2,i)
    %imshow(rgb2hsv(subI));
    HSVImage = rgb2hsv(subI); % Change to HSV
    intensity = HSVImage(:,:,3);
    map= (intensity>=0.9);
    percentage=sum(map(:))/numel(map)*100;
    P{i} = percentage; % get percentage of image that is above 0.9 intensity
end

% This essentially picks the image that is most white for the white
% background behind the number.

% this section could be easily changed to filter on a different
% condition or get rid of it all together to extract all text from
% all text boxes.

[~, idx] = max(cell2mat(P));
textBBoxes = textBBoxes(idx,:);
```

Figure 5: Code to be adapted to detect more numbers or numbers on a different background

As mentioned before, this only works well if properties about where the number is are known. An option if it is unknown is to get to the point where text boxes have been filtered on aspect ratio and stroke width and then use OCR to extract the text/numbers from all of the text boxes. This may provide a some noise in the output, though after the earlier filtering of text boxes it should be manageable.

It seems like there is a lot that can be adapted in this function, however to do so would require significant editing within the function to change how many areas of interest are being looked for or where the areas of interest are. Some of these options could be coded into the function if more variables were being taken initially like detectNum('IMG_0660.JPG','IntensityFilter',0.9,'NumberRegionInterest',3) could be looking for 3 text boxes with a white background with intensity greater than 0.9.

References

[1] https://uk.mathworks.com/help/nnet/ref/alexnet.html

[2] https://uk.mathworks.com/help/vision/examples/automatically-detect-and-recognize-text-in-natural-images.html