INM460 Computer Vision Coursework

Face Recognition and Object Character Recognition

Joaquin Perez-Lapillo
joaquin.perez-lapillo@city.ac.uk

---

1. Project overview and data preparation

This project report summarises the development of two functions in the context of Computer Vision systems, with the objective of gaining empirical experience in the techniques seen in lectures and tutorials. The first function was developed to detect and recognise individuals in an image given a set of labels, while the second detects the number a person is holding in an image or video. Both functions are interesting applications for a wide range of domains, such as cybersecurity and authentication.

The main technology used was Matlab R2018b with intensive use of the Computer Vision and Deep Learning Toolboxes, which proved to be highly efficient to develop the functions required for this project.

The functions were designed and tested on a dataset of images and videos provided specially for this coursework, containing 69 individuals who belong to the 2019 Computer Vision cohort at City, University of London (including the lecturer).

The face recognition function was tested against two groups of unseen images to account for its performance. When passing a set of holdout faces from individual images, all models showed almost perfect accuracy (99%). However, when tested against a labelled group image, the best model only achieved 57.1% accuracy. This shows the difficulty of predicting in “real world” conditions due to the models’ propensity to overfit the training data.

In the case of the OCR function, it was tested on 87 still-images performing considerably better than the face recognition function, with an accuracy of 72.4%.

Development process diagram

At a high-level, the face recognition process showed to be more extensive both in complexity and time spent for development. Prepare the database, extract features, and training the classification models took approximately 40 hours of work, while developing the OCR function took half of that.

The development of the face recognition function was planned in 5 stages: processing videos to increase the size of the dataset, get faces from images using the Matlab face detector, train the required classifiers with combination of feature extraction techniques, test them on individual and group pictures, and finally develop the function calling the pre-trained models (Figure 1).

On the other hand, the object character recognition function was designed in 3 stages: develop the single-image OCR function, evaluate its performance, and create a final “production” function that provides an answer for both videos and still images (Figure 2).

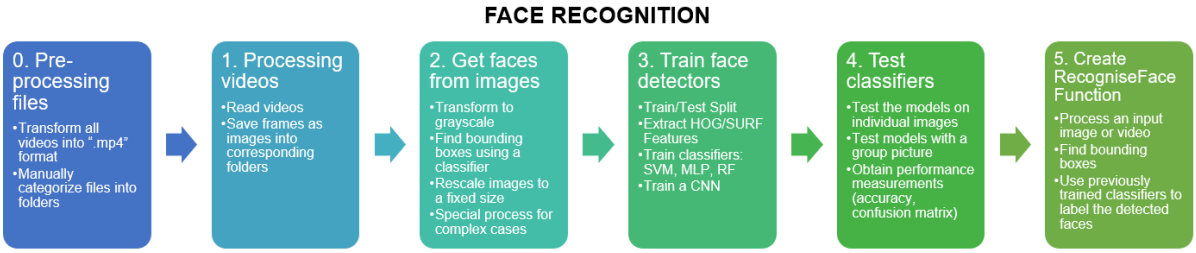

Figure 1: *Face recognition function development process*

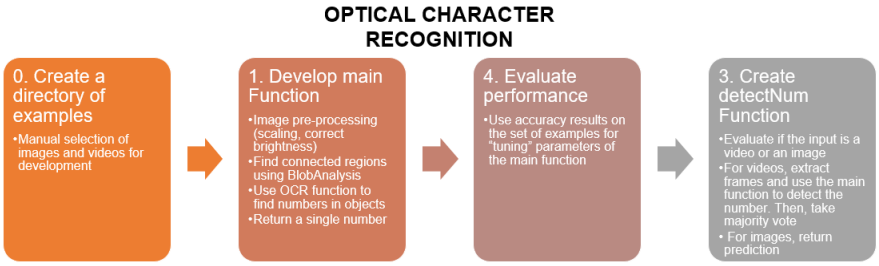

Figure 2: *Object character recognition function development process*

Preparation of the dataset

A first step that was key to increase the size of the training data was to convert images originally in ".heic" format to any other extensions readable for Matlab (I chose to transform all images to ".jpg"). This was done manually using iMazing HEIC converter. Same approach was taken for ".mov" videos, transforming them into ".mp4" files using the Videoproc app. Once images and videos were in the right formats, I manually assigned each individual file (image/video) to the corresponding folder following the number that they were holding (label).

For having a successful model training process for face recognition, Matlab expects to read images from structured directories, where each subfolder name corresponds to the label of the individual. Even though this was expected to be done automatically by creating a function that identifies the number on each image (object character recognition, second part of the project), I chose to do it manually because the OCR function proved not to be 100% accurate.

To increase the size of the dataset each video was processed using the VideoReader Matlab function[1]. The first 30 frames were extracted and saved into their corresponding folders, for each video. Thanks to this process, the final set of individual images per person was increased to an average of 130 (from originally an average of 4), giving a total of 9,662 images.

The next step was to create a database of faces, placing each in its corresponding labelled folder[2], using the 9,662 original images. For achieving that goal, the designed process iterated over each image using the build-in Matlab function CascadeObjectDetector[3] contained in the Computer Vision Toolbox. This function uses the Viola-Jones algorithm to detects parts of the body of people (Viola and Jones, 2001). After trying different combinations of classification models (FrontalFace CART and LBP, UpperBody, ProfileFace), and regularization parameters, I decided to do a first run on all images using only the FrontalFace CART[4] model, which showed to be the best, with a relatively large merge

[1] For details of the video processing, see Matlab script "**CV01_ProcessVideos.m"**

[2] Detailed script: "**CV02_GetFaces.m"**

[3] https://uk.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html

[4] Classification and regression tree analysis.

threshold of 10, for suppressing false detections. I also constrained the size of the faces to be between 100x100 and 700x700. The resulting faces were then resized to a standard of 100x100 pixels.

Running the code for all images for the first time resulted on an average of 90 faces identified per person, with some of them being far below that average (Figure 3). In general terms, it is interesting to notice that most of the poor results were associated with individuals with darker skin tones.

Given that obtaining a decent amount of faces was key to have enough data to train the classifiers, I decided to do a second run for the individuals below the average, now relaxing the regularization parameters to the Matlab default values.

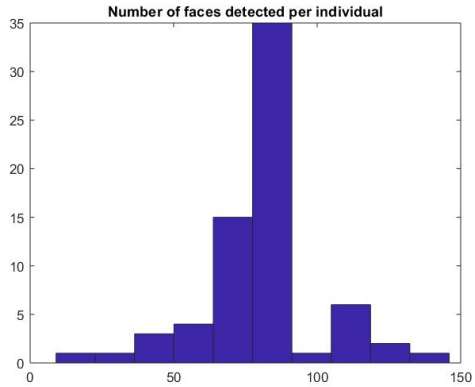

Figure 3: *histogram of faces detected per label before relaxing regularization parameters*

As expected, relaxing the regularization parameters increased the number of images detected to a minimum of 90 faces, but with a cost in terms of accuracy. An example of false positive detections is shown in Figure 4. In general, some shirts, ears, and arms were incorrectly detected as faces. All of them were manually deleted from the database.

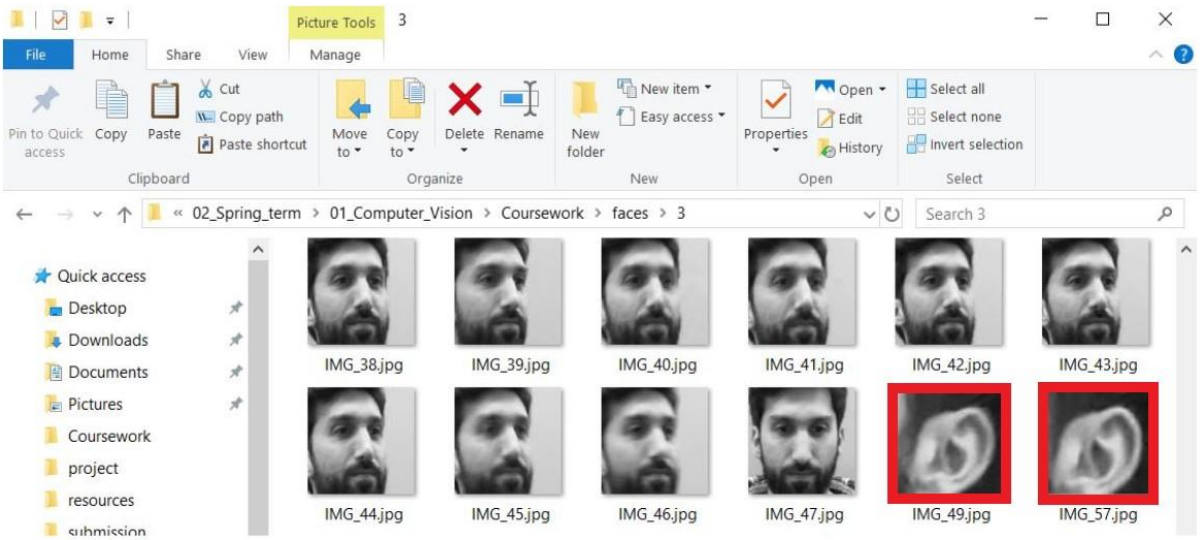

Figure 4: *false positive results of the face detection process marked with a red frame*

2. Face recognition

The face recognition function, called "**RecogniseFace.m**", was designed after training and evaluating a set of 6 models, with different feature types, using the database of 8,761 faces obtained in the data

preparation process. The list of classifiers included Support Vector Machine (SVM), Multilayer Perceptron (MLP), Random Forest (RF), and a Convolutional Neural Network (CNN), while the feature types used were Histogram of Gradients (HOG) and Speed Up Robust Features (SURF).

To make fair comparisons between the algorithm's performances, the full set of face images was reduced to 90 per person (6,210 in total), which was the minimum, and the separated into 70% for training (63 faces) and the rest for testing (27)[5]. An exception was made for the convolutional neural network where I kept the training size but separated the testing into validation and test (15% each). The main reason was to keep the same amount of training images for all classifiers.

The combination of feature types and classifier names trained (and hence valid combinations for testing the function) is shown in Table 1.

featureType	classifierName
'HOG'	'SVM'
'HOG'	'MLP'
'SURF'	'SVM'
'SURF'	'MLP'
'SURF'	'RF'
'NIL'	'CNN'

Table 1: *combinations of feature types and classifiers trained*

Feature extraction

Two types of feature extraction were tried for this project. The first one is called histogram of oriented gradients ("HOG") and it is used in Computer Vision mainly with the purpose of object detection. This technique counts the occurrences of gradient orientation in localized portions of the image, creating a histogram that represents each individual picture (Lecture notes, 2019). For the specific case of this application and given that the images were rescaled to a standard of 100x100 pixels, the HOG feature vector array reached to a size of 4,352 elements. In Matlab, the function for extracting this kind of features is ***extractHogFeatures***[6]***.***

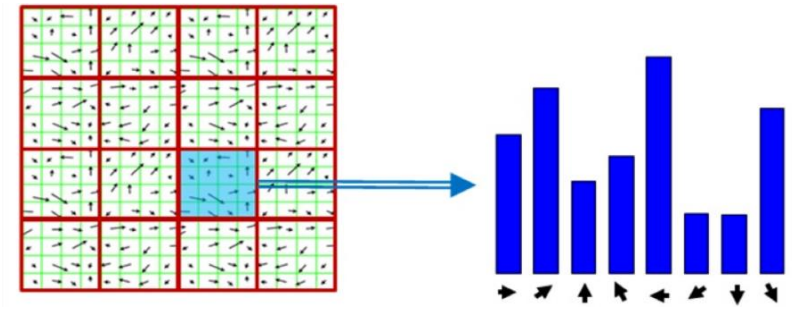

Figure 5: *HOG feature extraction process (from Lecture notes, session 5)*

The second method used for extract features from images was Speed Up Robust Features ("SURF"). This technique works as a local feature detector and descriptor that can be used for tasks such as object recognition, classification, or 3D reconstruction (Lecture notes, 2019). Based on another method called Scale Invariant Feature Transform (SIFT) descriptor, SURF is both faster and more robust against different image transformations than SIFT, keeping the benefits of a scale invariant feature extractor (Bay et al., 2006). The practical implementation of SURF was achieved by creating a bag of 500 visual words[7] of each image using the ***bagOfFeatures*** object in Matlab, which uses k-means

[5] "Single fold" approach.

[6] https://uk.mathworks.com/help/vision/ref/extracthogfeatures.html

[7] https://uk.mathworks.com/help/vision/ref/bagoffeatures.html

clustering to group the "visual words" together[8]. The result is a histogram of elements just like HOG (Figure 6), but the number of such elements is much narrower. The size of the bag can also be a hyperparameter, but I decided to use the Matlab default value for k due to time constrains.

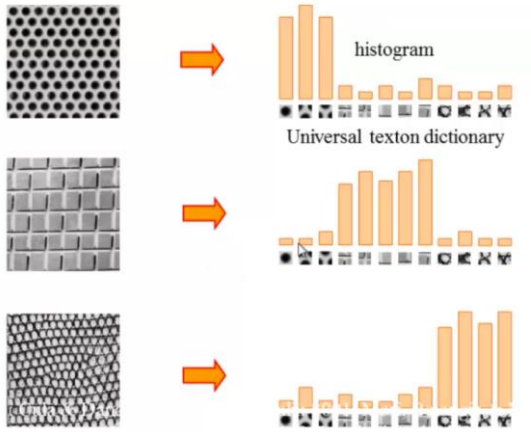

Figure 6: *bag of visual words extraction (from Lecture notes, session 5)*

Training classifiers

Four different classifiers were trained to compare their performance. In the next paragraphs I will describe each one briefly, providing also the mechanisms to implement them in Matlab.

Support Vector Machines (SVM)

It is a popular algorithm for supervised learning problems, programmed to classify data using a separating hyperplane between the target classes. The optimum hyperplane will be the one with the largest margin between the support vectors for each class. Those supporting vectors are observation closest to the hyperplane. SVMs use a kernel-based transformation approach for classes that cannot be linearly separated, with linear, radial basis function (RBF), and polynomial as alternatives (Burges, n.d.).

In practice, SVM was used by calling the ***trainImageCategoryClassifier*** function in Matlab[9] when SURF features were selected because it allows the input of the previously extracted bag of visual words. For the case of HOG features, SVM was used calling the ***fitcecoc*** function[10] which is not specific for images but for any kinds of data.

Multilayer perceptron (MLP)

It consists of a feedforward net, typically composed of neurons with unidirectional connections organised in one input layer, one or more hidden layers, and an output layer (Zanaty, 2012). The input layer represents the features vector, while the output layer provides a prediction given the signals that were received from the previous layers. The algorithm learns by sending the error back through the network and changing the weights accordingly in a process called backpropagation. Hence, the learning task is based on the minimization of the error function defined by the training examples, which leads to the optimal weights of the network.

For this project, the ***feedforwardnet*** Matlab function[11] was used with a hidden layer size of 100 neurons and scaled conjugate gradient backpropagation as training function ('*trainsgd*'), which trains any network if its weight, net input, and transfer functions have derivative functions. Backpropagation

[8] https://en.wikipedia.org/wiki/K-means_clustering
[9] https://uk.mathworks.com/help/vision/ref/trainimagecategoryclassifier.html
[10] https://uk.mathworks.com/help/stats/fitcecoc.html
[11] https://uk.mathworks.com/help/deeplearning/ref/feedforwardnet.html

is used to calculate derivatives of performance with respect to the weight and bias variables[12]. For both built MLP models (with HOG and SURF features), training stopped when the maximum validation failure rate was reached, i.e. when validation error did not decrease after 6 iterations. The choice of hidden layer size, training function and stopping criteria was due to lecturer's recommendations and, again, a grid search was not performed due to time constraints.

Random Forest (RF)

Combines a defined number of decision trees, all constructed by a different bootstrapped dataset using a subset of variables at each step (Braiman, 2001). The most voted option is chosen as output in classification problems (the average of the predictions for regression).

In the context of this project only one RF model was trained using the Matlab function ***TreeBagger***[13], using SURF features, and with 100 trees.

Convolutional Neural Network (CNN)

AlexNet[14] is a convolutional neural network that was previously trained on more than a million images from the ImageNet database to classify them into 1,000 labels[15]. The network has 25 layers (Figure 7), from which I extracted the last 3 highlighted layers corresponding to the original classification task (the 1,000-neurons fully connected layer, the softmax and the output layers). Those 3 layers were replaced by a 69-neurons fully connected layer, a softmax and an output layer ad-hoc to the 69 labels. Because Alexnet requires the use of 227x227x3 images, I adapted the code to use RGB images of the required size.

1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench' and 999 other classes

Figure 7: *Alexnet original network architecture and layers description*

After some experimentation with augmentation techniques, I decided to apply [-10 10] random rotation to the training set, together with random translation on the X and Y axis of [-3 3] and random

[12] https://uk.mathworks.com/help/deeplearning/ref/trainscg.html
[13] https://uk.mathworks.com/help/stats/treebagger.html
[14] https://uk.mathworks.com/help/deeplearning/ref/alexnet.html
[15] http://www.image-net.org/

reflection. As shown afterwards, this helped the algorithm to generalise better due to an increase in variance of the training images provided.

The training process took considerably more than the rest of the algorithms with a total of 82 minutes and 42 seconds (Figure 8). Running on a standard CPU, none of the above models took more than 10 minutes to build, showing the complexity involved in training a deep network.

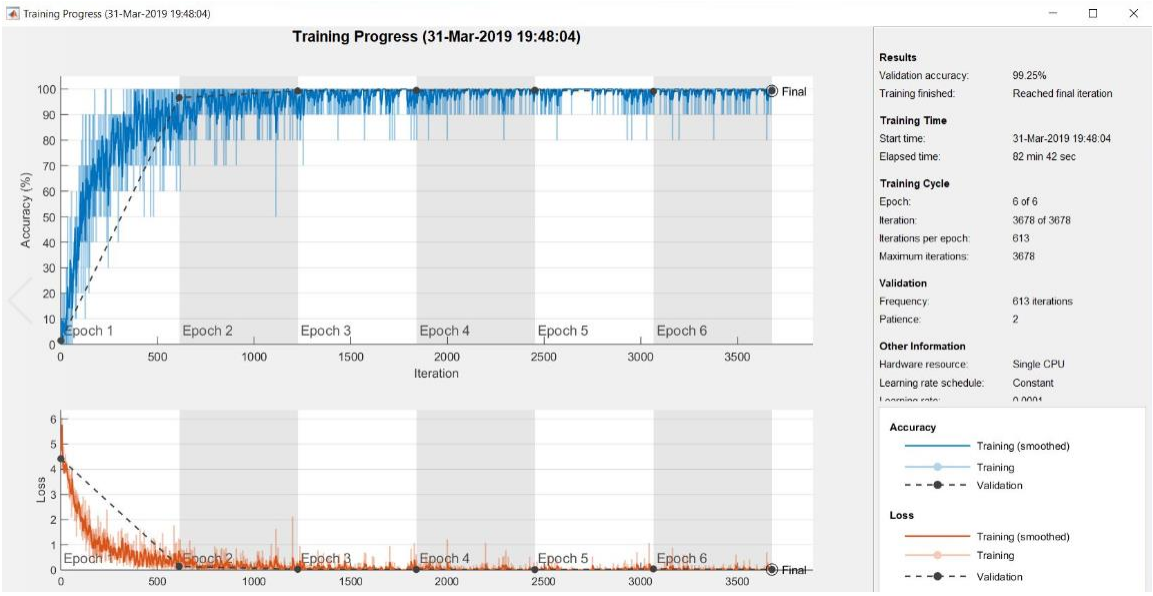

Figure 8: *CNN training progress. Plots of training accuracy and validation error over time*

Performance

Although each model's performance on training and test sets was recorded, I decided that was key to measure their performance against a group picture, so I manually labelled one selected image including 22 individuals ("IMG_8224.jpg") to use it as a "ground truth". As seen in Figure 9, accuracy results were surprisingly lower than results on the test set of faces that were randomly separated from the full faces dataset (Table 2). The best performing model was HOG_SVM (57.1%), followed by the CNN (52.4%).

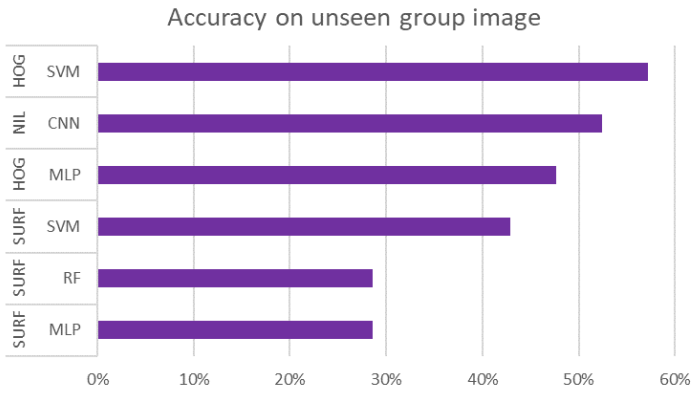

Figure 9: *Models accuracy on a selected test group image*

featureType	classifierName	trainingAccuracy	testAccuracy
HOG	SVM	100	99.8
HOG	MLP	99.9	99.7
SURF	SVM	100	99.7
SURF	MLP	99.9	99.7
SURF	RF	100	99.6
NIL	CNN	100	99.1

Table 2: *training and test accuracy per model on individual images*

This kind of behaviour normally suggests a lack of generalisation capabilities of the models, this is, they were not able to identify individuals in "real world" conditions because they did not see the faces in the same ways they are being presented.

It is no surprise that the best performing models, aside from the CNN, were the ones using HOG features because the size of the feature array was much extensive than for SURF (4,352 versus 500), allowing those models to find more specific patterns to recognise individuals.

The best model performance on the labelled group picture (HOG_SVM) is shown in Figure 10. Interestingly, it seems to be sensitive to the size of the faces, because more small faces were incorrectly classified, compared to bigger ones. This can also be due to occlusion and less-quality lightning conditions for the faces that are in the back of the room.

Figure 10: *HOG_SVM accuracy on group picture (ticks= correct, X=incorrect, ?=not in the dataset)*

When the HOG_SVM model was tested on individual images, it achieved almost perfect accuracy (99.8%). However, it still got some faces wrongly classified (Figure 11).

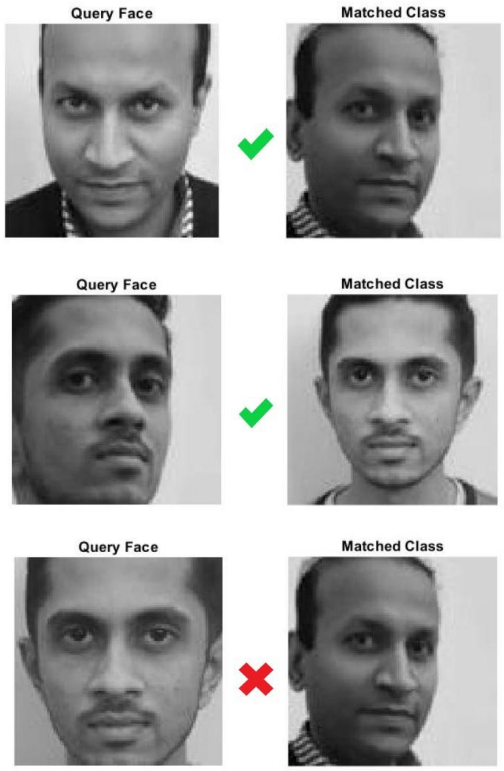

Figure 11: *HOG_SVM test accuracy on individual pictures*

Function structure and process

The function was designed to load and pre-process an image, detect the faces in it, load the queried classifier and feature type and perform classification to obtain their labels. Lastly, it retrieves a matrix including the individuals ID, centre-x position and centre-y position. This process is summarised in the pseudocode below and is detailed in Figure 12.

```
RecogniseFace.m
1. Read image
2. Transform to grayscale
3. Create a CascadeObjectDetector for Frontal Faces (CART), set parameters
4. Find BBOXes using the object detector
5. Count faces detected, initialise "P" matrix and set scaling constants
6. Load requested model (featureType, classifierName) and labels table
7. For each detected face
      Scale RGB/grayscale face according to each classifier requirements
      Extract HOG/SURF features from test face (if necessary)
      Predict label using pre-trained classifier
      Save label, x and y central face position to "P"
```

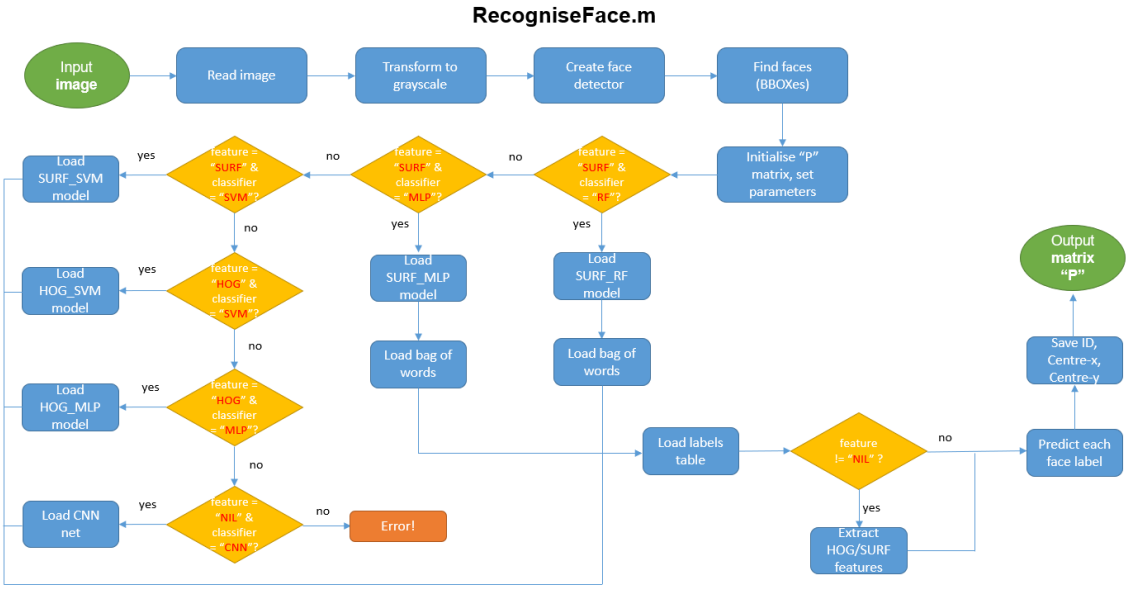

Figure 12: *RecogniseFace.m function block diagram*

3. Object character recognition

The OCR function was designed in two steps. First, a "main" function called **detectNumImg.m** was designed to work over individual images, using a developing set of 87 pictures[16]. The actual process is described in the following pseudocode:

```
detectNumImg.m
1. Create a BlobAnalysis object for finding connected regions
2. Resize, transform to grayscale and rotate image (if needed)
3. Increase brightness (if needed)
4. Transform to black and white (inverse) format
5. Find regions of interest (BBOXes) using the BlobAnalysis object
6. Filter relevant BBOXes: small, vertical rectangles on the centre
7. Increase size of the BBOXes to facilitate number detection
8. Detect numbers using OCR function
9. Filter objects where no numbers were detected
10. Filter objects with low-confidence detections
11. For each object detected
      Append number to an array
12. Return a single output value
```

In step 1, the function creates an object to identify connected regions using ***vision.BlobAnalysis***[17], which returns the properties of these regions of interest that are going to be helpful later to filter only the relevant ones for the problem. After applying some basic transformations to the image (resizing, transform to grayscale and rotating in case it is needed), I decided to increase the brightness of each image according to its mean brightness value, due to the presence of shadows on the paper area in some cases. Of course, if the image was already bright enough, the process did not change increased the brightness significantly. This was achieved by applying a linear function representing the relationship between original mean brightness and the increasing value required for correct the

[16] Experimental code is available in "**CV05_OCR.m**". The individual pictures were randomly selected from the original dataset.

[17] https://uk.mathworks.com/help/vision/ref/vision.blobanalysis-system-object.html

image[18]. The slope of the linear equation defined becomes a parameter that can be changed to achieve better results.

Because the BlobAnalysis object requires a binarized input image, the function transforms the grayscale image into the inverse of the black a white, which proved to work better than the regular form.

After finding the list of regions of interest (BBOXes), the next step is to filter the relevant ones. I do this by defining a minimum and maximum width of the box (between 10 and 50 pixels) as well as keeping only the vertical rectangular ones (width/height ratio between 0.3 and 1). I also filter the BBOXes that are outside the internal 90% area of the picture.

Because the BBOXes identify objects in a relatively tidy area, I increased each one by moving 2 pixels back in the width and height position and adding 4 pixels to its size.

The set of filtered BBOXes was then passed to the Matlab ***ocr***[19] function with the intention of recognise the text present in the regions of interest, ensuring that only numbers were identified[20]. This function not only returns the text that has being recognised but also the confidence level of the prediction. I found relevant to filter the objects that were retrieved with low levels of confidence, so a threshold was set to accomplish a better prediction. The best performing threshold was a dynamic, depending on the highest confidence level found on the image. The idea is to retrieve two numbers only if they have a relatively similar confidence level (the second number retrieved should have at least 80% of the confidence found for the highest one).

Table 3 shows the results of some experiments run to adjust the function to the best parameters to achieve better performance, using the 87 images selected for development. The best combination of parameters reached to 72.4% accuracy in recognising the number each person is holding. However, when we extend the measure of accuracy giving half of the credit to partial detections (detect at least one number), the accuracy achieved for the best implementation is 80.5%.

increase box size	max lenght	detection confidence	brightness slope	accuracy	accuracy extended
4	50	67%	-1	58.6%	70.7%
3	50	80%	-1	63.2%	73.6%
3	50	67%	-0.95	64.4%	76.4%
3	50	67%	-0.9	64.4%	76.4%
3	50	70%	-1	66.7%	76.4%
3	50	67%	-1	67.8%	77.6%
3	40	67%	-1	67.8%	77.6%
3	30	67%	-1	67.8%	77.6%
2	50	67%	-1	70.1%	78.7%
2	**50**	**80%*Max**	**-1**	**72.4%**	**80.5%**

Table 3: *parameter tuning process for OCR function*

The first image in Figure 13 (from left to right) shows an example of a correctly recognised number ("33") even though some BBOXes that did not correspond to numbers passed the filters (hand and eye). The example in the middle shows a partial detection because the function only retrieved "3" as an answer, and the last one shows a fail of the function. In general, it was noticed that the function does not perform well if the picture is not centred (like in this last example) because of the alteration of the numbers' shapes.

[18] Change in brightness = 255 -1*(mean brightness)

[19] https://uk.mathworks.com/help/vision/ref/ocr.html

[20] 'CharacterSet' was limited to '0123456789'.

Figure 13: *examples of correctly recognised number, partial recognition and fail to recognise*

Finally, the **detectNum.m** function reads both images and videos and uses the previous function to provide an answer. The only relevant contribution of this function is processing the cases when the input file is a video. It reads the file, extract 30 frames from it[21], passes the image to the previous function to obtain an answer, and calculates the majority vote between all frames. For still-images, it just returns the answer of the main function. This process is summarised in the pseudocode:

```
detectNum.m
1. Check if the input file is an image or a video
2. If video
      Read video
      Extract frames
      For each frame
            Use detectNumImg.m to detect the number in the image
            Save output to an array
      Take majority vote and return output
   Else
      Read image
      Use detectNumImg.m to detect the number in the image
      Return output
```

It is important to mention that **detectNum.m** only accepts input images in the formats *.jpg and *.jpeg, and videos in *.mp4 and *.mov formats.

4. Discussion

First, it is interesting to mention that all models trained for the face recognition function showed great ability to predict on images coming from the same set as the training, this is, similar faces without occlusion or lightning problems. However, when testing them on an unseen group picture, they did not perform as good as before. This behaviour shows the need for more variance in the training set, increasing the training set size, applying intentional blurry techniques, rotation, and alterations. The idea is to recreate as much as possible the "real world" conditions of group pictures.

[21] It reads frames 10 to 40 because initial and final frames of videos tend to be blurry.

The histogram of gradients (HOG) features showed to be more efficient in terms of prediction capabilities compared to the SURF features, but it is necessary to say that the size of the array was larger in the case of HOG. In future, I would like to extend this project by using a more similar array length. In principle, SURF features should outperform HOG features considering that they are scale invariant.

Finding that the CNN was not the best performing model was also interesting considering that it involves building a much complex model than the rest in terms of the number of parameters, taking also several times longer to train. We must consider also that the CNN model was the only one including augmentation of the training images (rotation, translation, and reflection). A probable cause for this relatively poor performance could be the necessity of more training data to reach the CNN's potential as a classifier.

Future work in this project should also include hyperparameter tuning for all machine learning models. For example, trying different kernels for SVM (gaussian or polynomial), different hidden layer sizes, training functions, and stopping criteria for MLP, and iterating over numbers of trees for RF.

For the CNN I would also like to explore the use of a dropout layer to prevent overfitting, which seems to help in similar cases (Srivastava et al., n.d.).

Adding an extra label to the problem would be also interesting so that the models can classify unseen faces (not included in the original dataset) as an additional class, e. g. "0" or "99", instead of providing the most look alike person as an answer. Additionally, it would be interesting to extend the code to provide a prediction of the emotion of each individual face (happy, sad, or neutral).

In relation to how to evaluate the model's performance, this project should be perfected by testing the classifiers on more group images, to have a more robust idea of which one is providing more stable results.

In the case of the OCR function, I would like to continue finding better ways to filter the regions of interest that are relevant for the project and try some of the previously trained deep networks for recognising the digits instead of the Matlab OCR function. Another interesting extension would be adapting the code to find more than one set of numbers in the image.

References

Burges, C.J.C., n.d. A Tutorial on Support Vector Machines for Pattern Recognition. SUPPORT VECTOR Mach. 47.

Braiman, 2001. "Random Forest", Machine Learning Journal, 45(1), 2001, pp. 5-32.

Lecture notes, 2019. Computer Vision module, City, University of London (Dr. Sepehr Jalali)

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., n.d. Dropout: A Simple Way to Prevent Neural Networks from Overfitting 30.

Zanaty, E.A., 2012. Support Vector Machines (SVMs) versus Multilayer Perception (MLP) in data classification. Egypt. Inform. J. 13, 177–183. https://doi.org/10.1016/j.eij.2012.08.002