INM460 Computer Vision Coursework

Face Recognition and OCR

Heiko Maerz heiko.maerz@city.ac.uk

Link to Google Drive folder:
https://drive.google.com/drive/folders/1emStOn7tCcPc6azaZkN1Na5mVlqGmtnF?usp=sharing

1 Introduction

This report covers the coursework for the elective course INM460 Computer Vision taught at City, University of London in the spring term 2019 by Dr. Sepehr Jalali and Dr. Gregory Slabaugh.
The project includes two implementation tasks:

1. A face recognition task to detect and identify (via an identification number) the faces of the lecturer and a subset of students from the cohort;
2. An optical character recognition task (OCR) to detect and recognise a number on a piece of paper that a person is holding in an image or video. This number is also used as the label for the face recognition task.

Both tasks belong to the domain computer vision, an area with a number of applications, such as biometric access control, security use of CCTV, but also in consumer areas such as tagging of image libraries on social media.

The programming environment used for this task is MatLab version 2018b including the installation of the Computer Vision System Toolbox, Image Processing Toolbox, Optimization Toolbox, Neural Networks and Statistics, Machine Learning Toolbox and the Parallel Computing Toolbox.

The project is broken down into the following tasks:

1. Image preparation: download, pre-processing, and labelling.
2. Implementation of face recognition.
3. Implementation of OCR.

2 Image Preparation

The process steps for image preparation are shown in Figure 1

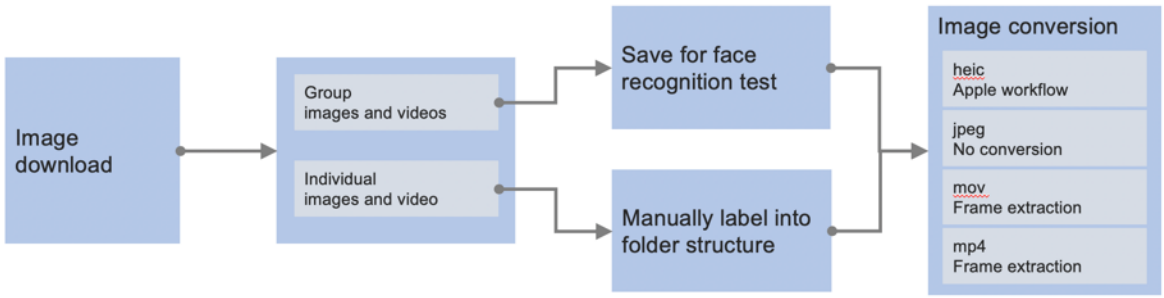

Figure 1 Image Preparation Process

2.1 Image Download

The images were provided for download via Moodle. They were taken by four cameras and include individual photos and videos of the instructor and 68 students of the cohort, group photos, and group videos.

2.2 Test and Training Images, Labelling

The downloaded data was split into 'individual' and 'group' images and videos.
Group data was saved in a test directory and will be used as unseen data to test the trained classifiers. Individual images and videos were manually labelled using the number each person is holding. The OCR function was developed at this stage but proved to be not accurate enough for the task. Each image or video was saved into a directory, the directory name corresponds to the label. This allows the use of imageSet[1] and imageDatastore[2]. These are image collections implemented in MatLab which hold image filename, label, and number of images in each label.

2.3 Image Conversion

The file formats of the data are of type

- .HEIC (High Efficiency Image File Format, a format for digital images and digital image sequences)[3],
- .jpeg/.jpg (lossy compression codec for digital images)[4],
- .mov (MPEG-4 compression codec for digital videos)[5],
- .mp4 (MPEG-4 compression codec for digital videos).

All images were converted into the jpeg format for further processing in the course of the implementation of this coursework.

- Images type jpeg did not require additional transformation.
- Images type HEIC were converted using a macOS Automator[6] ConvertHeicToJPG.workflow.
- All frames of videos of both type MOV and MP4 were extracted using the MatLab program ExtractVideoFrames.m. Some videos faded to black frames at the end of their sequence, a brightness was used threshold to stop the extraction of these.

3 Face Recognition

3.1 Requirements

This task comprised the development of a MatLab function that will return a matrix with the location of face(s) and the corresponding IDs for a given photo.
The face recognition should implement different feature types and classifiers.

The optional task of detecting a person's emotional state was not implemented during this project.

This project will use the feature types HOG and SURF, the machine learning algorithms MLP and SVM, it will also use AlexNet as an example of a CNN using transfer learning. A short overview follows.

3.2 Feature Types

Attributes or features, that is numerically measurable characteristics, are required to be able to process images for machine learning and classification. This project uses both Histogram of Oriented Gradients (HOG) and Speed Up Robust Features (SURF).

---

[1] https://uk.mathworks.com/help/vision/ref/imageset-class.html

[2] https://uk.mathworks.com/help/matlab/ref/matlab.io.datastore.imagedatastore.html?searchHighlight=imageDatastore&s_tid=doc_srchtitle

[3] https://en.wikipedia.org/wiki/High_Efficiency_Image_File_Format

[4] https://en.wikipedia.org/wiki/JPEG

[5] https://en.wikipedia.org/wiki/MPEG-4

[6] https://support.apple.com/en-gb/guide/automator/create-a-workflow-aut7cac58839/mac

3.2.1 HOG

The HOG is used in computer vision to extract shape features that describe silhouette and contour information. (Lecture slides 5). This approach divides the image into blocks with overlap, and each block into cells. The gradients in each of these regions are computed and the orientations are binned. The result is a histogram of gradients that can be used as a feature vector for an individual image. This process is depicted in Figure 2 (Source: Slides for Lecture 05).
The HOG vector size depends on the cell size, the block size, the overlap, the number of bins, and the image dimensions.

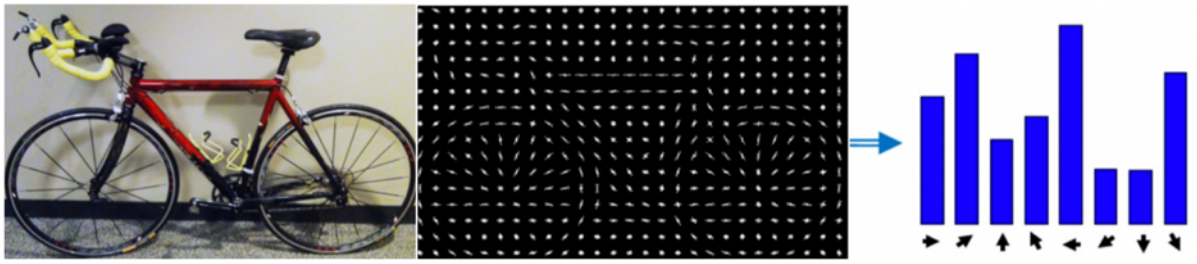

Figure 2 Histogram of Oriented Gradients, Source: Slides for Lecture 5

3.2.2 SURF

SURF is an alternative way to encode image features. It is a fast and robust algorithm to extract scale invariant features and was proposed by Bay, Ess, Tuytelaars, and Van Gool as an alternative to SIFT. The algorithm extracts a bag of features or 'visual words' and then uses k-means to cluster these.

3.3 Classifiers

Two different classifiers are implemented in this project, Multilayer Perceptron (MLP) and Support Vector Machine (SVM).

3.3.1 MLP

MLP is a neural network algorithm used for supervised learning. The network consists of one input layer with a number of neurons corresponding to the input vector size, one or more hidden layers with a defined number of neurons, and one output layer. MLP is a feedforward architecture, each neuron is unidirectionally connected to every neuron in the next layer.
The neurons have an activation function and a bias, and each connection to a preceding neuron is associated with a weight. The output of a neuron is determined by its bias and the input values from each connected neuron multiplied by their corresponding weights. The output of the neuron is the activation function of the sum of these numbers. Weights and biases are parameters determined during training. The training process is back propagation, and the weights and biases are adjusted by sending the error (training output vs. training label) back through the network using a specific error function and training function.
The number of hidden layers, the number of neurons in each hidden layer, the activation function, and the training and error functions are hyperparameters which can be tuned to improve the performance of the model. (Zanatay, 2012).

3.3.2 SVM

SVM is a machine learning algorithm that uses a hyperplane (Figure 3[7]) to separate observations from the different target classes for classification. A support vector for each class is the data point that is closest to the separating hyperplane (Burges, C.J.C., n.d.).
If the classes cannot be linearly separated SVMs use kernel transformations. This is a hyperparameter that can be tuned to improve the model, however for this project the MatLab default was used.

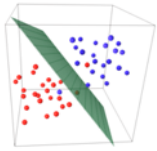

Figure 3 SVM Hyperplane

3.3.3 Convolutional Neural Network – AlexNet

This project uses the MatLab implementation of AlexNet. AlexNet was originally developed by Alex Krizhevsky, when published in 2012 it significantly outperformed all available image classification systems of its time.
AlexNet has 25 layers, and is pre-trained, which means the parameters are already optimised on the features for a large number of images[8]. The first 22 layers are copied, only the last three layers responsible for this specific classification have to be retrained (Figure 4): the generic AlexNet classifies 1000 different images, this implementation has 69 target labels. This process is called transfer-learning.

Ly.	Name	Type	Generic Matlab Implementation	Project Implementation
1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element	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]	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]	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element	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]	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]	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer	4096 fully connected layer
18	'relu6'	ReLU	ReLU	ReLU
19	'drop6'	Dropout	50% dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer	4096 fully connected layer
21	'relu7'	ReLU	ReLU	ReLU
22	'drop7'	Dropout	50% dropout	50% dropout
23	'fc'	Fully Connected	**1000 fully connected layer**	**69 fully connected layer**
24	'softmax'	Softmax	softmax	softmax
25	'classoutput'	Classification Output	crossentropyex with 'tench' and 999 other classes	crossentropyex with '1' and 68 other classes

Figure 4 Generic Matlab AlexNet, Recognise Faces Implementation

[7] Source: https://appliedmachinelearning.blog/2017/03/09/understanding-support-vector-machines-a-primer/

[8] https://uk.mathworks.com/help/deeplearning/examples/transfer-learning-using-alexnet.html

3.4 Implementation: Training Classifiers

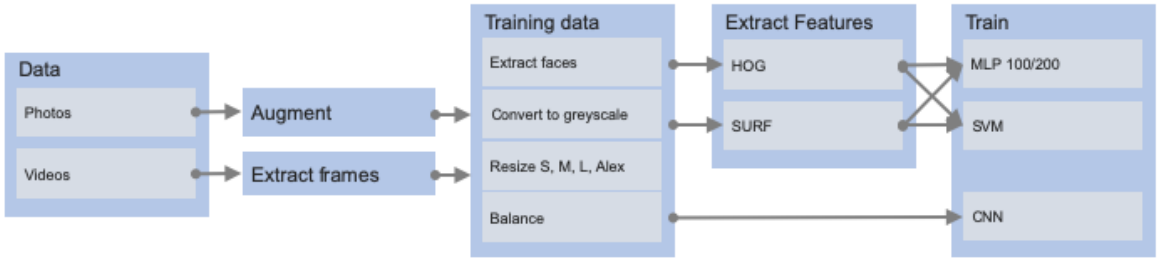

Figure 5 Training Face Recognition Classifiers

Figure 5 provides a high-level overview over the model training process:
The training data is obtained by augmenting the photos and extracting video frames. Faces are then extracted from these images and prepared for training: they are converted to greyscale, resized, and the number of training images are then balanced for each target class. SURF and HOG features are extracted for the SVM and MLP, and the models trained. The CNN trains directly on the extracted, and balanced image datastore.

The final test will be a group photo of the cohort. The faces in the front of the group photo are of bigger than the faces in the back, one experiment carried out in this project is to use training images of different dimensions to test how well a model trained on bigger training images generalises on small faces in the back compared to how well a model trained on smaller training images generalises on large faces in the foreground.

All programs, function, and data files are listed in the appendix at the end of this document.

3.4.1 Image Augmentation

The training and validation data are extracted from both photographic images (on average four per individual / class) and videos, which provide a large number of frames. To counter this imbalance a 'naïve' image augmenter (AugmentTrainingImages.m) was used on the photos to create additional training data: the photo is rotated -8, -4, 4, and 8 degrees, a blurred copy is also saved to add noise to prevent overfitting (Figure 6).

Figure 6 Rotated, Original, and Rotated and Blurred

3.4.2 Face Extraction

In the next step faces are extracted with program ExtractTrainingFaces.m. The faces from both photos and video frames are saved in a directory structure that reflects the class labels. The program uses the provided MatLab function vision.CascadeObjectDetector[9] which function uses the Viola-Jones algorithm to detect body parts.

Three different detection models are used in sequence using function fct_extract_training_face.m for the CascadeObjectDetector:

1. FrontalFaceCart: This detector uses 'Classification and Regression Tree analysis' to detect upright and forward facing faces using Haar features. If a face is detected it is saved in a directory corresponding to the class label and the next image is processed, otherwise

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

2. FrontalFaceLBP: This detector uses local binary patterns to detect upright and forward-facing faces. If a face is detected it is saved in a directory corresponding to the class label and the next image is processed, otherwise
3. ProfileFace: This detector uses Haar features to detect upright face profiles. If a face is detected it is saved in a directory corresponding to the class label and the next image is processed, otherwise an error is displayed in the MatLab command window

Experimenting with the CascadeObjectDetector showed that using a MergeThreshold of 8 resulted in the best compromise between detecting a face at all and returning a crop that is not a face. For some groups of images this parameter setting did not return a face, and the images had to be re-processed with a lower threshold. This resulted in false positives, which were deleted manually (Figure 7).
Restraining both minimum and maximum image size aided in the face detection, due to different resolutions for photos and video frames different sizes are used in both scenarios.

Figure 7 Face Detection: False Positives

At the end this processing step each class has a differing number of training images from photos and video frames. To be able to train classifiers without introducing bias the number of images was normalised, which results in each class having a total of 176 training images, 14 from the photos and 162 from the video frames.

As a last step in the face extraction process each training image is converted to greyscale, resized to the dimensions 75*75, 95*95, and 115*115 respectively for SVM and MLP and 227*227 RGB for AlexNet. The training images are saved into a folder structure /train/[Small/Medium/Large/Alex]/[class label] and are ready for model training.

3.4.3 Train Classifiers

Five combinations of feature types and classifiers are trained for this project:

- HOG and SVM (for three sets of training image dimensions: 75*75, 95*95, 115*115);
- HOG and MLP (for three sets of training image dimensions: 75*75, 95*95, 115*115, and two different network architectures: one hidden layer with 100 neurons and one hidden layer with 200 neurons);
- SURF and SVM (for three sets of training image dimensions: 75*75, 95*95, 115*115);
- SURF and MLP (for three sets of training image dimensions: 75*75, 95*95, 115*115 and two different network architectures: one hidden layer with 100 neurons and one hidden layer with 200 neurons);
- CNN for transfer learning: AlexNet.

The program TrainClassifiers.m is used to train the classification models. SVM and MLP are trained with a number of parameters, for reusability and efficiency this happens in functions fct_train_HOG_MLP.m, fct_train_HOG_SVM.m, fct_train_SURF_MLP.m, and fct_train_SURF_SVM.m respectively.

The training phase does not use cross validation. The datasets for MLP and SVM are split into 80% training, 20% validation. The dataset for CNN is split into 80% training, 10% validation, and 10% test. This ensures that all three classifiers are trained on the same number of images.

Each trained model is saved in a .mat file including the imageClassifier, the dimensions for image resizing, feature type attributes necessary for encoding (feature vector size for HOG, bag of visual words for SURF), classification labels, and model, classifier, and feature type name.

The HOG feature extraction is based on Lab 05 for the computer vision lecture. This project uses the MatLab function extractHOGFeatures() and the MatLab default values[10], but the attribute vector size is

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

different for each of the selected resolutions used in training the classifiers. Attribute vector sizes are 2304 for image dimension 75x75, 3600 for 95x95, and 6084 for 115x115.

The SURF feature extraction is based on Lab 05 for the computer vision lecture. This project uses the MatLab function bagOfFeatures() to define the bag of features, the vocabulary size of visual words was increased from the MatLab default of 500 to (rather arbitrarily) 1536 to be comparable in dimension to the HOG features. The images for model training were encoded using MatLab encode.

MLP training is based on Lab 05 for the computer vision lecture. The MatLab functions feedforwardnet(), configure(), and train()[11] are used to train each MLP. The default MatLab parameters are used, only the training function and the number of neurons in the hidden layer is specified.
This project uses one hidden layer with two different sizes: 100 and 200 hidden neurons. The training function is 'Scaled Conjugate Gradient', a function that employs second derivatives to determine (local) error minima more efficiently. The error function is mean squared error. Training was stopped when the maximum validation error rate was reached (early stopping).
Target-labels are one-hot encoded.

SVM training is based on Lab 05 for the computer vision lecture. The model is trained using the MatLab fitcecoc() function for multi-class classification using the default parameters.

Transfer learning is a direct adaptation of the MatLab tutorial "Transfer Learning Using AlexNet"[12]. An image data augmenter is part of the MatLab template, the aim is to add noise and variance to the training dataset to prevent overfitting. The resulting network architecture is shown in Figure 4, column "Project Implementation".

3.4.4 Test Classifiers

Train and validation accuracy were captured during training (Table 1). The performance was relatively high, with an average of 99% for training and 95% for validation. The real test however was to measure the performance against unseen data. For that purpose, a group image was manually labelled to get the 'ground truth'. Each classifier was tested against this data. The findings are discussed in section 5.

The resulting accuracies are listed in Table 1. Performances of selected models for the test image are displayed in Figure 8 (HOG and MLP), Figure 9 (SURF and MLP), and Figure 10 (SVM with HOG and SURF).

Feature Type	Classifier	Training Image Dimension	Train Accuracy	Validation Accuracy	Test Accuracy
HOG	MLP[100]	75x75	99%	97%	48%
HOG	MLP[100]	95x95	99%	98%	50%
HOG	MLP[100]	115x115	99%	95%	36%
HOG	MLP[200]	75x75	99%	98%	53%
HOG	MLP[200]	95x95	99%	97%	41%
HOG	MLP[200]	115x115	99%	97%	55%
HOG	SVM	75x75	100%	98%	50%
HOG	SVM	95x95	100%	97%	51%
HOG	SVM	115x115	100%	97%	53%
SURF	MLP[100]	75x75	99%	92%	15%
SURF	MLP[100]	95x95	97%	82%	8%
SURF	MLP[100]	115x115	98%	87%	29%

[11] https://uk.mathworks.com/help/deeplearning/ref/feedforwardnet.html

[12] https://uk.mathworks.com/help/deeplearning/examples/transfer-learning-using-alexnet.html

SURF	MLP[200]	75x75	99%	94%	29%
SURF	MLP[200]	95x95	99%	91%	32%
SURF	MLP[200]	115x115	99%	97%	36%
SURF	SVM	75x75	100%	93%	29%
SURF	SVM	95x95	100%	95%	31%
SURF	SVM	115x115	100%	96%	32%
CNN: AlexNet for 227x227 RGB			99%	99%	17%

Table 1 Train, Validation, and Test Accuracy

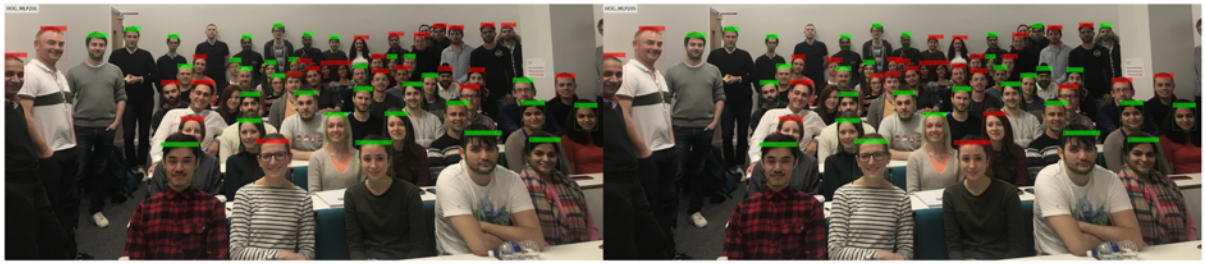

Figure 8 HOG MLP[200], Large and Small Training Images

Figure 8 visualises the performance of the combination of HOG and MLP. Green bounding boxes show correctly detected faces, red bounding boxes show wrong classification. The best performing model for the large (115x115, left) and the small (75x75, right) training datasets are shown. In each case the MLP with one hidden layer with 200 neurons performed best in test.

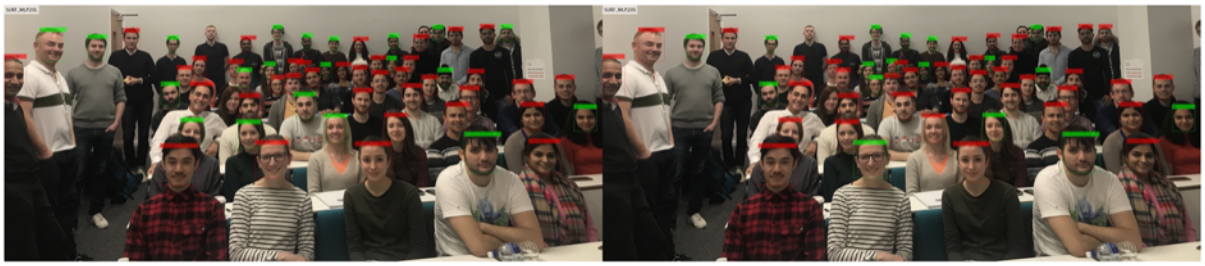

Figure 9 SURF MLP[200], Large and Small Training Images

Figure 9 visualises the performance of the combination of HOG and SURF. The best performing models for the large (115x115, left) and the small (75x75, right) training datasets are shown.

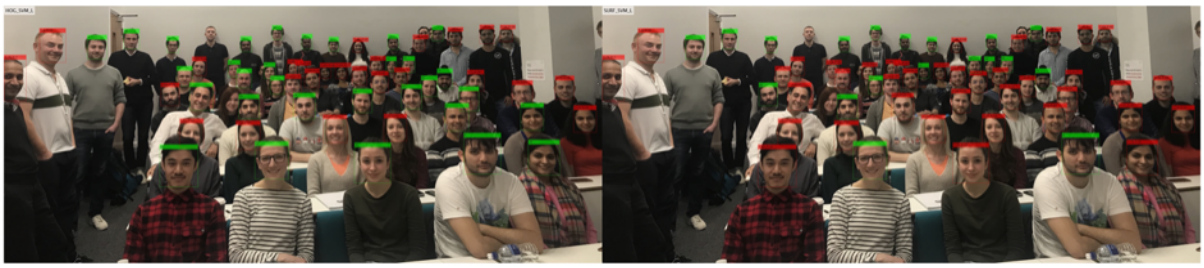

Figure 10 SVM:HOG and SURF

Figure 10 displays the performance of the best performing model for the combination of SVM with HOG (left) and SURF (right).

3.4.5 RecogniseFace

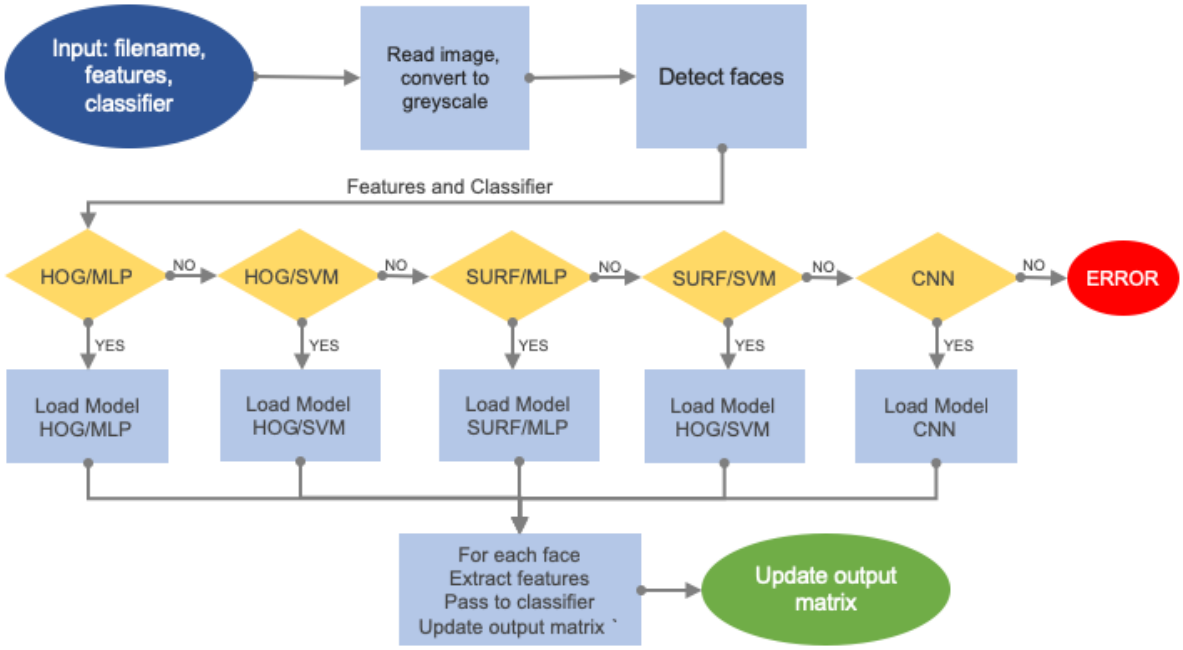

Figure 11 Flowchart RecogniseFace

The function

```
P = RecogniseFace(imageFilename, featureType, classifierName)
```

Is used to detect faces in images.

It accepts as inputs:

- imageFilename : the filename of the image, of type jpeg (or jpg)
- featureType : attribute feature type, valid inputs ['HOG', 'SURF', 'NONE']
- classifierName: classifier name, valid inputs ['CNN', 'MLP', 'SVM']

it produces the output:

- Matrix P (Id, face centre X, face centre Y)

The following combination of feature type and classifier are supported:

```
RecogniseFace({image filename}, 'HOG', 'MLP');
RecogniseFace({image filename}, 'HOG', 'SVM');
RecogniseFace({image filename}, 'SURF', 'MLP');
RecogniseFace({image filename}, 'SURF', 'SVM');
RecogniseFace({image filename}, 'NONE', 'CNN');
```

Figure 11 provides a high-level overview over the process steps:
The function is called with the parameters filename, feature type, and classifier name. The image is loaded and converted to greyscale. The faces are detected using vision.CascadeObjectDetector() with the FrontalFaceCart parameter, the MergeThreshold is set to 8.
Given the input parameters feature type and classifier name the model is loaded.
In a loop each face is resized according to the model's requirement, the features are extracted, and the class label is predicted. The ouput matrix is updated with the ID and X and Y coordinates of the central region of the face.

4 OCR

This task comprised the development of a MatLab function that accepts as an input the file name of an image or video of a person holding a number in their hand. This function should detect the number and return the value.

The optional task of detecting more than one number was not implemented during this project.

The actual optical character recognition is performed using the MatLab function ocr(). A quick experiment showed however that passing the raw image does not yield satisfactory results.

The challenge was to only pass the relevant part of the image that contains the number into the function, and this implementation is one attempt at doing so.

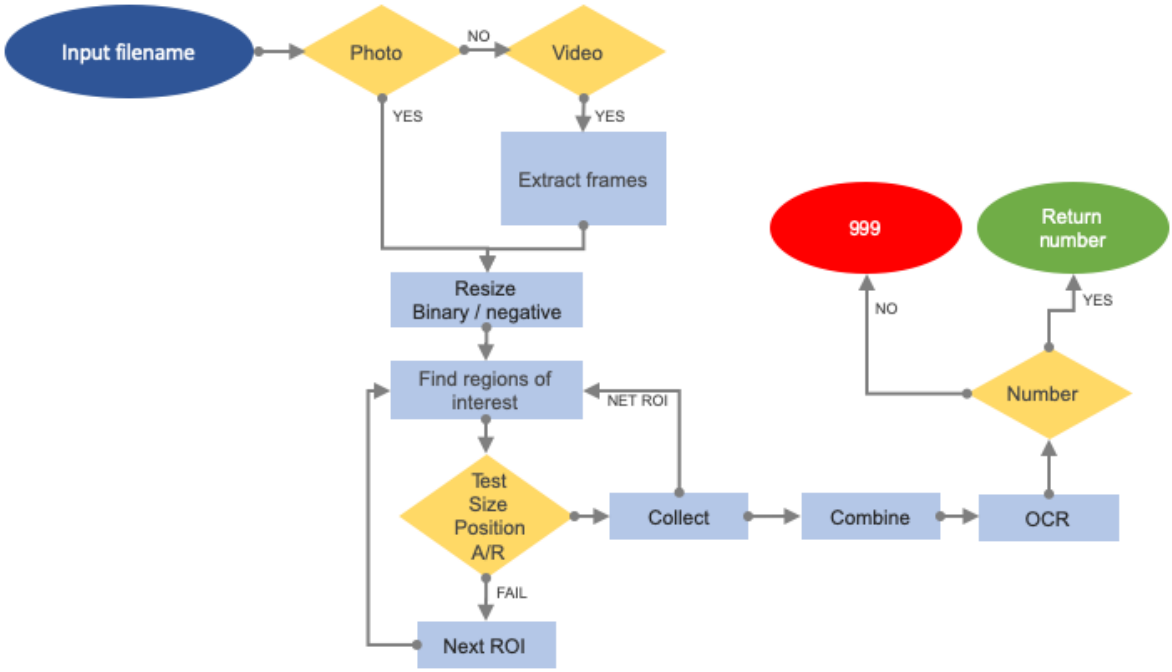

Figure 12 Flowchart OCR

Figure 12 provides a high-level overview over the implemented function detectNum(). The lower part displays image conversion, extraction of regions of interest, filter and combine regions of interest, and OCR. This process is described in detail below.

The approach chosen was to use MatLabs vision.BlobAnalysis() to extract regions of interest. This function returns statistics for connected regions in an image, and these connected regions are used in this program to find the number in the photo.

To speed up processing the image is first resized to a width in the neighbourhood of 720 pixels. This results in file that both preserves enough details yet is small enough not to strain computing resources. A known image dimensions is used later on when size is used as a criterion to select the part of the image that may contain a number.

The BlobAnalysis works best on binary images, images are converted from greyscale to binary black and white using a threshold. During development the assumption was made that the number to be detected would be printed in black ink on a white piece of paper. As such the paper should be relatively bright, the number relatively dark, given the overall brightness of the image. Experiments have shown that the images are not uniformly lit, and a fixed value for the threshold proved to be of little effectiveness. The threshold to convert to binary ‘black’ or ‘white’ was therefore set to the average brightness of the entire image.

Further experiments have shown that BlobAnalysis() return regions of interest with higher consistency that are white on black rather than black on white. The binary image passed into the function is therefore a negative image.

BlobAnalysis returns a number of regions of interest, and for the task not all are, well, actually of interest. Based on the assumption that all images tested for OCR will follow a pattern a number of criteria are applied. All these criteria where derived empirically by experimenting with different images and numbers to find appropriate limits and thresholds.

- The region of interest should be in 'portrait' orientation, that is the height should be bigger than the width.
- Numbers are of interest here and experimenting with examples has shown that each number returns its own region of interest. For that reason, the aspect ratio of height to width is used to select regions that might contain a number. This number may be a 'skinny' 1 or a more robust '8' for example. This project will use height / width < 4.2.
- Since the image is of known size further assumption can be made about size, that is minimum and maximum height, width, and area.
- One more assumption about the images used for OCR is that the number will not be at any border, i.e. top, bottom, left, or right. Any region of interest that touches image borders is ignored.

The result of this processing step is a matrix with X and Y coordinates for the top left anchor point, the X and Y coordinates of the central area of the image, and width and height. It is of note that the area has been enlarged by 4 pixels in all direction to not only capture the potential number, but a bit of 'background' to facilitate OCR.

As previously noted each number creates its own region of interest, however this function should be able to return double digit numbers as well. This necessitates combining regions of interest depending on their proximity. The employed process is thus:
All regions of interest are sorted according to their position from left to right. The program loops through all regions. For each region, all regions to the right of it are compared in regard to the position of their central point. If the distance falls within a certain threshold (determined by the size of the region) the two areas are merged.

Figure 13 Image Pre-Processing for OCR

This process is illustrated in Figure 13, which shows from left to right a resized greyscale image, the negative binary image passed into the blob analyser, two bounding boxes as the result of the first filter, and both boxes combined due to their proximity. It can be noted that the checked shirt returned a number of regions of interest which were filtered out due to size and area.

The image and the bounding boxes that are the result of the previous processing steps are then passed into the MatLab ocr() function. The ocr function is restricted to only recognise numbers by setting the parameter CharacterSet to numerical chars. In case that more than one bounding box returns a number the ocr result with the highest confidence is returned for this image. If no number is detected the function returns 999.

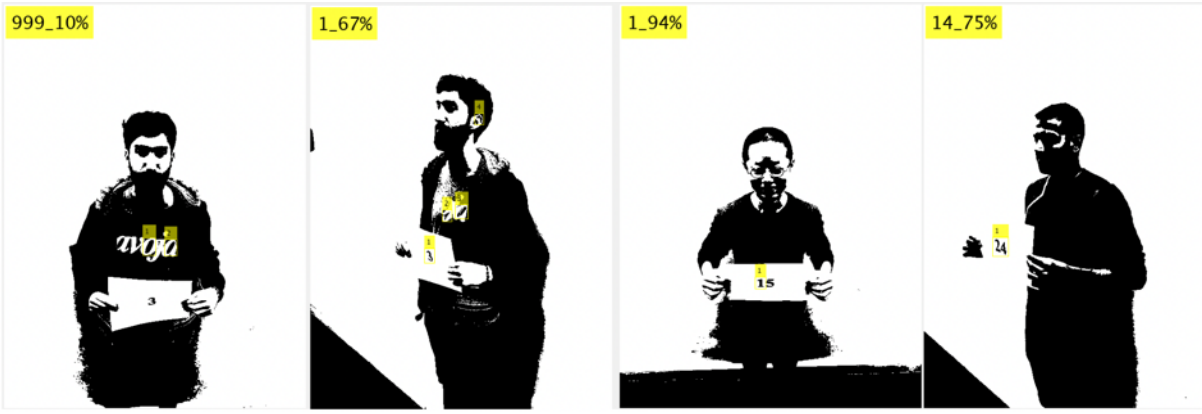

Figure 14 OCR Errors

Figure 14 shows examples in which this approach fails. Each example is annotated with the number detected and the corresponding OCR confidence. From left to right the first example failed to return a bounding box containing the number at all but did identify a few other areas. The function returns the default error number 999. The next example shows a number of bounding boxes, one of which contains the number. The number however is not recognised correctly. In the third example only one number was captured in a bounding box. This number was identified correctly, but the second digit is missing altogether. The last example has captured both numbers in one bounding box, the OCR function however recognised only the second digit correctly.

4.1 detectNum

The function

```
ocrNum = detectNum(filename)
```

is called to detect a number in an image or video.

function detectNum(filename)

It accepts as an input :

- filename of an image or video: the file extension of an image must start with a j (eg .jpeg), the file extension of a video must start with a m (eg .mov)

It produces an an output

- the number that the person is holding; the function will return 999 if the number could not be detected.

The flowchart for this function is depicted in Figure 12:
The image or video is loaded, in case of a photo the data is processed directly, in case of a video the frames are extracted.
The image or video frames are then passed to function

```
[ocrNumber, ocrConfidence] = fct_img_ocr(imgIn)
```

Which performs the process described in section 4.

5 Discussion

5.1 Face Recognition

All models returned very good training and evaluation performance. When tested against actual group pictures, which is the goal, all models returned very poor performance.

The quality of the training images and of the actual group photos differs quite a bit, both in regard to resolution and lighting. Detected faces from the group images are smaller (especially in the back), lit poorer, and have less contrast. Occlusion is an issue as well, with many faces half blocked.
Nevertheless, it has to be noted that the models trained in this project overfit and generalise very poorly. The images used for testing do require more variance for better real-world accuracy, using a wider variety of facial angles and tilts, frontal-facing and profiles, as well as rotation and blur. Comparing results with other students in the cohort suggested reducing the number of training images to stop overfitting, but a quick experiment in my setup with a total of 76 training images per class resulted in even worse accuracies.
For future work I would like to experiment more with the approaches and methods to select proper training images, and to further investigate the benefits of image augmentation.

In general models using HOG perform significantly better than models using SURF. On the one hand this is not surprising, because the feature vector size for SURF is 1536 while it is 2304, 3600, and 6084 for the small, medium, and large training images for HOG respectively. On the other hand SURF should be more robust in regards to different angles and tilted faces. This is another area worth investigating for future work.

In regard to the size of the hidden layer it can be noted that 200 neurons perform better than 100 neurons, especially in case of the largest HOG feature vector, where the layer size of 100 struggles with an input vector of size 6084.

SVM and MLP perform reasonably similar, the models submitted using HOG for this project have shown test accuracies of 55% for MLP[200] and 53% for SVM. The best performing model for the project is an MLP with one hidden layer and 200 neurons trained on HOG features extracted from large dimension training images.

The test accuracy for the CNN is surprisingly poor. At this time I do not quite understand it, as future work I would like to investigate the reason for this.

Experimenting with different architectures and training image dimensions did not yield interesting insights to improve model accuracy. There is not pattern of smaller faces (in the background) being detected better by the model trained on smaller images and bigger faces (in the foreground) being detected better by larger images.
For future work I would therefore like to investigate how changes to the training image dataset can improve the model performance, especially in regard to methods for selection of adequate training images, optimum training set volumes, approaches to ensure variance in the training set, and training image augmentation.

5.2 OCR

The OCR development was characterised by experimentation to determine good parameters for the detection of regions of interest and merging of areas. The submitted approach will reflect on the lack of computer science background of the author, because there surely are more efficient ways to approach the task at hand.

Here again the performance was disappointingly poor. For future work I would like to investigate how to improve the quality of the area containing the number to be passed on to the OCR function. Special interest is of finding points of reference to un-skew images taken form an angle, such as the second example in Figure 14.

5.3 Concluding Thoughts

An interesting extension to this work would be the implementation of the optional task for face recognition, i.e. also detect the emotional state, as well as implementing a model to perform OCR.

References

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

IN3060/INM460 Computer Vision 2019, City, University of London Lecture Notes and Lab Sessions, Dr. Sepehr Jalali and Dr. Gregory Slabaugh, available at https://moodle.city.ac.uk/course/view.php?id=30973

MatLab Help as cited in the footnotes, available at https://uk.mathworks.com/help/matlab/

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

Appendix

Files for this project, bold files are required for task 2 (face recognition) and task 3 (OCR)

Step	Program Name	Description
Face Recognition	**RecogniseFace**	**Function to recognise faces in an image**
Face Recognition	**fc_ALEX.mat**	**Trained classifier CNN**
Face Recognition	**fc_HOG_MLP.mat**	**Trained classifier HOG MLP**
Face Recognition	**fc_HOG_SVM.mat**	**Trained classifier HOG SVM**
Face Recognition	**fc_SURF_MLP.mat**	**Trained classifier SURF MLP**
Face Recognition	**fc_SURF_SVM.mat**	**Trained classifier SURF SVM**
OCR	**detectNum**	**Function to detect a number in a photo or video**
OCR	**fct_img_ocr**	**OCR function used inside detectNum**
Image Conversion	ConvertHeicToJPG.workflow	MacOs conversion workflow HEIC → JPEG
Image Conversion	ExtractVideoFrames.m	Extract video frames
Train Image Classifier	AugmentTrainingImages.m	Rptate (-8, -4, 4, and 4 degrees) and blurr photos
Train Image Classifier	ExtractTrainingFaces.m	Extract training images from photos and videos
Train Image Classifier	fct_extract_training_face.m	Use classifier and image as input to return a face
Train Image Classifier	TrainClassifiers.m	Train actual classifiers
Train Image Classifier	fct_train_HOG_MLP.m	Function to train classifier
Train Image Classifier	fct_train_HOG_SVM.m	Function to train classifier
Train Image Classifier	fct_train_SURF_MLP.m	Function to train classifier
Train Image Classifier	fct_train_SURF_SVM.m	Function to train classifier
Test Image Classifier	SetupTestFaceDetectors.m	Program to determine 'ground truth' for test
Test Image Classifier	TestFaceDetectors.m	Program to test all trained classifierss
Test Image Classifier	fctTestFaceClassifier.m	Function to test an individual classifier