INM460 Coursework Report

Tim Laibacher
City, University of London

April 10, 2018

1 Introduction

This report provides an overview of the methods and approaches used in order to fulfill the coursework requirements. The report is structured into five sections: Section 2 explores the dataset. Section 3 outlines the methods and approaches used to build the face detection and classification function "RecogniseFace". Similarly Section 4 discusses methods and approaches used to implement the OCR function "detectNum" that detects the number written on a white A4 paper that is held by a person. Section 5 concludes and explores the strengths and weaknesses of the implementation and scope for further improvement.

2 Dataset

The dataset consists of 53 folders that contain mug shots of individuals, holding a white A4 paper with an identification number printed on it, from different perspective along with short movie sequences that were recorded right before and after the photo was taken. An additional folder contains group photos of all individuals (or a large fraction of them). The number of raw images and videos per individual is shown in Figure 1. The difference in file endings, "JPG" vs "jpg", is likely due to way photos were imported from different devices. Additional we noticed that all photos taken with an iPhone SE were in horizontal orientation, which we addressed via a function that checks the exif information and rotates the photos if necessary.

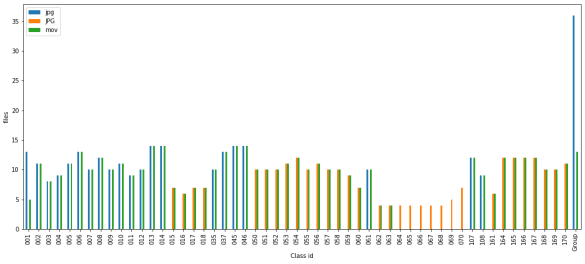

Figure 1: Images and videos per class id in the raw dataset

3 Face detection and classification

This section introduces key concepts and methods (Section 3.1) and outlines the implementation details of the "RecoginiceFace" function (Section 3.2).

3.1 Methods

In this section we are going to introduce the theoretical concepts and methods that will be used in the "RecogniseFace" function. First we will explore the methods for the detection of one or multiple faces in an image (Section 3.1.1), followed by the techniques used for the classification of individual faces to one of individual ids (classes) - also referred to as face recognition (Section 3.1.2).

3.1.1 Face detection

3.1.1.1 Histogram of Oriented Gradients (HOG)

The HOG feature descriptor, introduced by Dalal and Triggs [1], can be used in combination with

classification methods, such as SVM or MMOD, to detect objects in images. It divides the input image with a fixed aspect ratio of $1:2$ into smaller blocks of size 16×16 with a 50% overlap. Each such block consists of 2×2 cells with size 8×8. For each of the 8×8 cells a histogram of gradients is calculated, whereby the gradient orientation is quantized into 9 bins (0-180 degrees). The direction of the gradient determines which bin, and the magnitude of the gradient determines the vote. The final feature vector is created by concatenating all histograms.

3.1.1.2 Max-Margin Object Detection (MMOD)

In comparison to traditional cascade classifier, which rely on a set of positive and negative images and therefore perform sub-sampling and do not make efficient use of the available training data, MMOD introduced by King [2] is a SVM based method that does not require such sub-sampling. MMOD proofed to have higher accuracy on the public FDDB face detection dataset [3] compared to the well-known Viola-Jones method [2]. The objective function is defined as:

$$\begin{aligned} & min_{w,\xi} \frac{1}{2}||w||^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \\ \text{s.t. } & F(x_i, y_i) \geq max[F(x_i, y_i) + \triangle(y, y_i)] - \xi_i, \forall i \\ & \xi_i > 0, \forall i \end{aligned} \tag{1}$$

The loss is defined as

$$\begin{aligned} \triangle(y, y_i) = L_{miss}(\text{num. of missed detections}) + \\ L_{fa}(\text{num. of false alarms}) \end{aligned} \tag{2}$$

where L_{miss} and L_{fa} control the relative importance of achieving high recall and precision and C is equivalent to the C parameter in conventional SVM (see Section 3.1.2.5).

3.1.2 Face classification

3.1.2.1 ResNet

Until the introduction of the ResNet architecture, conventional Convolutional Networks, such as [4] and [5], regularly achieved state-of-the-art results on popular image classification tasks, such as ImageNet [6]. Training deeper and deeper networks

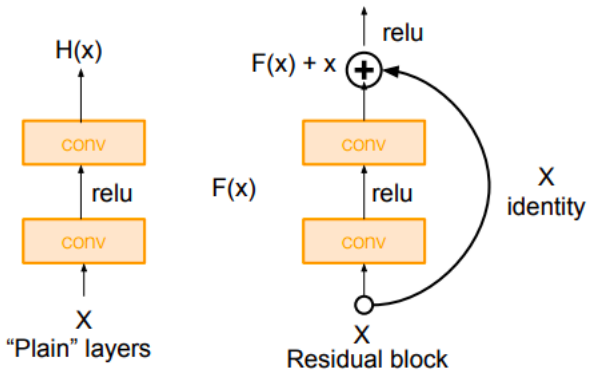

Figure 2: Residual block, adapted from [8]

however eventually does not necessary lead to higher training accuracy, but instead the accuracy saturates and then degrades [7]. To address this issue, He, Zhang, Ren, *et al.* [7] introduce residual blocks as shown on the right side of Figure 2. In comparison to "plain" layers that directly fit the desired underlying mapping, residual blocks use a pass-through identity mapping, which allows the network to fit a residual $F(x) = H(x) - x$ that is the difference between the value before the block and the value after the block. To prevent gradients from vanishing, ReLus are used as activation functions. Additionally batch normalization takes place after each convolution. The final architecture consists of multiple stacked residual blocks, a per-channel global average pooling layer as penultimate layer and a final fully connected layer. He, Zhang, Ren, *et al.* [7] introduce five variants with 18, 34 ,50, 101 and 152 layers respectively. Variants with more than 34 layers have additional "bottleneck" layers in each residual block to increase efficiency. Our implementation uses the 34 layer variant as shown in Figure 14. It is light weight - the trained model is 85 mb in size - and thereby leaving enough room within the 200mb submission limit for source-code, alternative models and sample images and videos. Before settling with the ResNet34 architecture, we also experimented with ResNet18 and ResNet50. ResNet18 achieved roughly 2% worse accuracy compared to ResNet34. ResNet50 surprisingly was roughly on par with the performance of ResNet34 but came with an increase in model size that would exceed the submission file size limit.

3.1.2.2 Scale-Invariant Feature Transform (SIFT)

The SIFT algorithm invented by Lowe [9] aims to create highly distinctive features that are invariant to image scale, rotation, illumination and 3D camera viewpoint. These properties make it a good candidate for image classification, where the extracted features are used as an input for classification algorithms, such as the Support Vector Machine and Logistic Regression variants used in this report. The SIFT algorithm consists of four main stages, which we will explore in the following paragraphs.

1. Scale-space Extrema Detection In order to detect keypoints (corners) of different scales, scale-space filtering by the differences-of-Gaussians (DoG) as an approximation for Laplacian of Gaussian is used. DoG is defined as the difference of two Gaussian blurrings of an image, with $k\sigma$ and σ that are convolved with the image I:

$$D(x,y,\sigma) = G(x,y,k\sigma) - G(x,y,\sigma) * I(x,y)$$
$$\text{with} \quad G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} exp(\frac{-(x^2+y^2)}{2\sigma^2}) \tag{3}$$

To calculate $D(x,y,\sigma)$, the paper proposes the method depicted in Figure 3: For different octaves of the image pyramid, the image is convolved with Gaussians to produce images separated by a constant factor k in the scale space (left column). The resulting Gaussian images are then subtracted by their adjacent neighbors to generate the DoGs shown in the right column. The Gaussian image is down-sampled by a factor of 2 after each octave. The resulting DoGs are then searched for local maximum and minimum, where each pixel in the image is compared with it's eight closest neighbors and nine closest neighbors in the preceding and following scale as shown in Figure 4.

2. Keypoint Localization This stage refines the keypoint candidates that were calculated in the previous step by using the Taylor series expansion of the scale space to increase the accuracy of the extrema locations and a 2×2 Hessian matrix with a threshold of 10 to remove keypoints located on edges.

3. Orientation Assignment In order to make the detected keypoints invariance to image rotation, an orientation is assigned to it. To do so, the gradient magnitude and direction of a selected neighborhood is calculated. An orientation histogram with 36 bins covering 360 degrees is created from the gradient magnitudes. The highest peak and any peak above 80% of the histogram is used to create keypoints. Multiple keypoints with different directions can therefore be created at the same location.

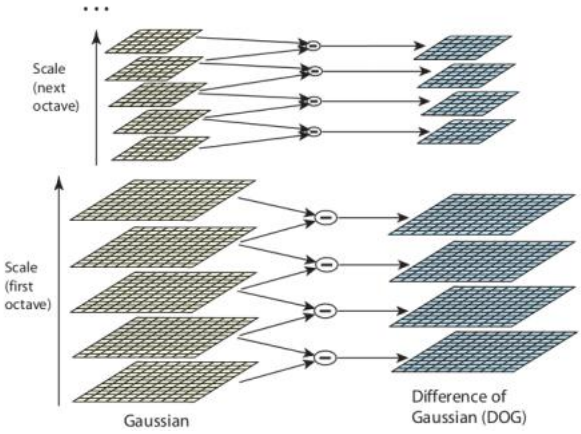

Figure 3: Scale-Space Pyramid with Differences of Gaussian, adapted from [9]

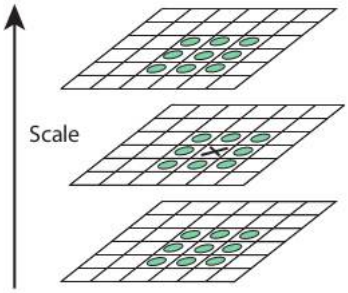

Figure 4: Local extrema of the DoG, adapted from [9]

4. Keypoint Descriptor A 16×16 sample around each keypoint is devided into 16 sub-blocks of size 4×4. For each 4×4 block a orientation histogram with eight bins is created, resulting in $8 \times 4 \times 4 = 128$ descriptor values per keypoint.

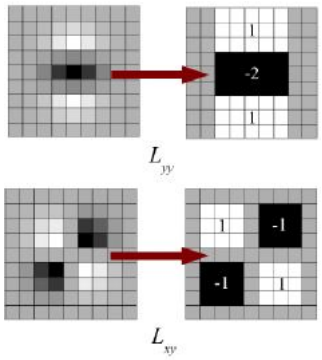

Figure 5: Box Filter as an approximation of the Gaussian second order partial derivative, adapted from [10]

3.1.2.3 Speeded-Up Robust Features (SURF)

SURF can be seen as a speeded up version of SIFT. As the key concepts are similar, this section only highlights the key differences between the two methods.

Instead of using DoG as an approximation of the Laplacian of Gaussian to build the image pyramid, SURF uses an integer approximation to the determinant of Hessian matrix for scale and location.

$$Hess(x,\sigma) = \begin{bmatrix} \mathcal{L}_{xx}(\boldsymbol{x},\sigma) & \mathcal{L}_{xy}(\boldsymbol{x},\sigma) \\ \mathcal{L}_{xy}(\boldsymbol{x},\sigma) & \mathcal{L}_{yy}(\boldsymbol{x},\sigma) \end{bmatrix} \quad (4)$$
$$det(Hess_aprox) = D_x x - D_y y - (0.9D_{xy})^2$$

where D_{xx}, D_{yy} and D_{xy} are the Box Filter approximations for the Laplacians. Figure 5 illustrates such an approximation. This is advantageous as convolution with box filter can be evaluated very fast with the use of integral images.

As shown in Figure 6, while SIFT uses different scales of the image, in SURF different scales of Gaussian masks are used. The scale of image is therefore not altered in SURF, which decreases the computational costs further.

The last difference between SURF and SIFT lies in the way the orientation and the descriptors are found. Here SURF uses the sum of the Haar-wavelet response around the keypoint. To calculate the orientation, Haar-wavelet responses are evaluated within a neighbourhood of radius $6s$ where s is the scale at which the point was detected. The orientation with the largest sum of horizontal and

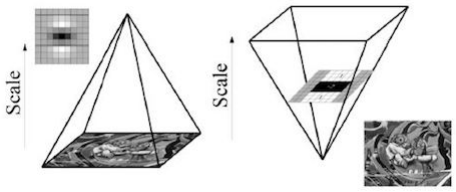

Figure 6: Instead of iteratively reducing the image size in (SIFT,left) , the use of integral images allows the up-scaling of the filter at constant cost (SURF,right), adapted from [10]

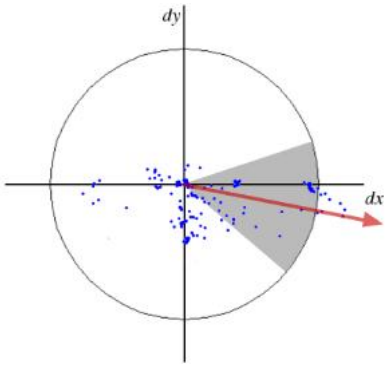

Figure 7: Orientation assignment via sliding orientation window, adapted from [10]

vertical wavelet responses is calculated by a sliding orientation window of size $\frac{\pi}{3}$, illustrated in Figure 7.

To extract the descriptor (Figure 8), a square region of size $20s \times 20s$ is taken around the keypoint. This region is further divided into sub-blocks of size 4×4. For each sub-block horizontal and vertical wavelet-responses are computed from 5×5 samples. The feature vector is formulated by summation of the x and y responses and their absolute values:

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|) \quad (5)$$

This is done for all 4×4 subregions, resulting in a final feature vector of size $4 \times 4 \times 4 = 64$. We can see that while SIFT generates a 128 dimensional feature vector, SURF default output has only half the size.

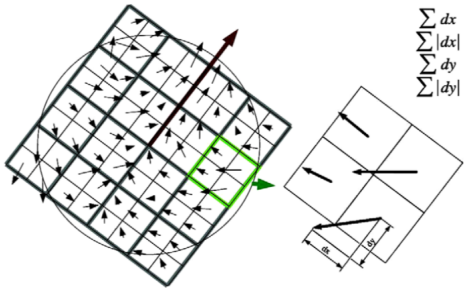

Figure 8: Calculation of descriptors, samples have size 2×2 instead of $5\times$ for illustration purpose, adapted from [10]

3.1.2.4 K-Means

K-Means is a popular clustering algorithm, given a data set with of N observations of a random variable $\boldsymbol{x}$, the algorithm partitions it into K clusters. The goal is to minimize the following objective function:

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||x_n - \mu_k||^2 \tag{6}$$

1. Choose initial values for μ_k, the centroids of the kth cluster.
2. Minimize J w.r.t. r_{nk}, the set of binary indicator variables.

$$r_{nk} = \begin{cases} 1, & \text{if } k = argmin_j||x_n - \mu_j||^2. \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

3. Minimize J w.r.t. μ_k

$$\mu_k = \frac{\sum_n r_{nk}x_n}{\sum_n r_{nk}} \tag{8}$$

4. Repeat 2 and 3 until convergence

[11]. We will make use of K-Means to create a vocabulary of visual words (codebook) that is used in the bag-of-visual-feature classification approach.

3.1.2.5 Support Vector Machines (SVM)

Given two classes with training vectors $x_i \in \mathbb{R}$,$i = 1, ..., l$ and a vector $y_i \in \{1, -1\}$, Suppert Vector Classification solves the following primal problem:

$$\begin{aligned} &min_{w,b,\xi}\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \\ &\text{subject to } y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i \end{aligned} \tag{9}$$

where $\phi(\boldsymbol{x}_i)$ maps $\boldsymbol{x}_i$ into a higher-dimensional space and $C > 0$ is the regularization parameter. The dual problem is solved as it lends it self more naturally to the inclusion of the kernel function, necessary when dealing with feature spaces that are not linearly separable:

$$\begin{aligned} min_\alpha \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{y}^T\boldsymbol{\alpha} = 0, \\ & 0 \leq \alpha_i \leq C, \quad i = 1, ..., l, \end{aligned} \tag{10}$$

where $e = [1, ..., 1]^T$ and $C > 0$ is the upper bound, Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv y_i y_k K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and the kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$. Solving (10), the optimal $\boldsymbol{w}$ is:

$$\boldsymbol{w} = \sum_{i=1}^{l} y_i\alpha_i\phi(\boldsymbol{x}_i) \tag{11}$$

and the decision function:

$$\begin{aligned} &sgn(\boldsymbol{w}^T\phi(\boldsymbol{x}) + b \\ &= sgn(\sum_{i=1}^{l} y_i\alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b) \end{aligned} \tag{12}$$

As a kernel function we used the Radial Basis Function (Gaussian) of the form:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma||\boldsymbol{x}_i - \boldsymbol{x}_j||^2} \tag{13}$$

[12]. As a consequence two hyper-parameters have to be defined: C and γ. Multi-class classification via SVM can be done in a one-vs-one strategy. This method trains separate binary classifier for each different pair of classes, leading to $\frac{N(N1)}{2}$ classifiers.

3.1.2.6 Logistic Regression (LR)

The L2 penalized binary logistic regression model minimizes the following cost function:

$$min_{w,c}\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n} log(exp(-y_i(\boldsymbol{x}_i^T\boldsymbol{w} + c)) + 1) \tag{14}$$

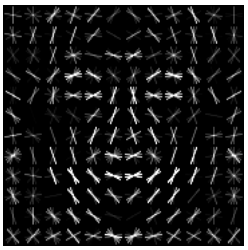

Figure 9: Learned HOG face detector, adapted from [16]

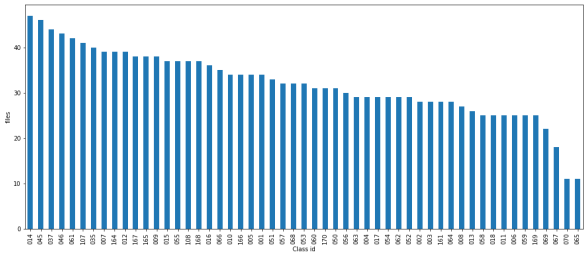

Figure 10: Images per individual (class id) in the "much face" dataset

[13]. Leaving us with only one parameter C that needs to be defined. In order to solve a multi-class classification problem the logistic regression optimization problem can be decomposed in a "one-vs-all" fashion, where separate binary classifiers are trained for all classes.

3.2 Implementation of the "RecogniseFace" function

The "RecogniseFace" function pipes several building blocks discussed in the previous section together. On a high-level there are two phases: face detection and face recognition. The following sections cover both phases and the blocks they are made out of.

3.2.1 Face detection

We used a version of the MMOD detector, introduced in Section 3.1.1.2, that was pre-trained on the Labeled Faces in the Wild dataset [14] using HOG features. This version is available in the dlib machine learning toolkit [15]. An illustration of the resulting hog detector is shown in Figure 9. It was considered to include a CNN face detector, however the computational costs on CPU inference mode were deemed to high and as a consequence is disabled by default. The source-code still contains this options for potential future work.

3.2.2 Face Classification

In order to classify the extracted faces, we trained three classifiers outlined in Section 3.1.2: SVM, Logistic Regression and ResNet34. For SVM and Logistic Regression two models for each classifier were trained, one with SIFT features and one with SURF features, resulting in 5 models in total.

3.2.2.1 Dataset creation

Before we could train our models, a dataset of images that contain only individual faces had to be created. First individual faces were extracted from mug shots and group photos using the face detection method introduced in Section 3.2.1. For individuals where this resulted in less than 25 faces, additional images were extracted from the available video files. For four such individuals (ids:065,070,067,069) no video was available as highlighted in Figure 10. The resulting dataset, named "much face", contains 1679 face images. A train-test split of 80%-20% was used to evaluate the performance of our models.

In addition to this dataset, an extended version, "much face plus" was created in order to account for the nine persons that are in the group photos but do not have individual photos and an id assigned. It contains 1868 face images for 62 individuals.

3.2.2.2 Training of the ResNet34 classifier

We used a ResNet34 model pre-trained on the ImageNet dataset as fixed feature extractor. The last fully connected layer was reset and it's output size changed to 53 and 62 for the "much face" and "much face plus" datasets respectively. The Input images were resized and center-cropped to 224×224 and normalized with standard-deviation and mean of 0.5 for each channel. We selected cross-entropy as the loss function and stochastic gradient descent with an initial learning rate of 0.001 and a mo-

mentum of 0.9 as optimizer. The learning rate was further decayed every 7 steps by a factor of 0.1. Training was stopped after 24 epochs and the total training time on a Paperspace [17] GPU+ instance with one NVIDIA Quadro P4000 was 7 minutes.

3.2.2.3 Training of the bag-of-visual-words classifier

The training phase of each of the four bag-of-visual words classifier (SVM + SURF, SVM + SIFT, Logistic Regression + SURF, Logistic Regression + SIFT) consists of four steps as shown in Figure 11.

1. *Convert the image to grayscale and extract features (SURF or SIFT descriptors).*

2. *Create a visual vocabulary (codebook) via K-Means clustering.* Each found cluster center μ_k becomes a code-vector. We used a vocabulary size of $k = 500$, resulting in 500 code-vectors. An initially tested vocabulary of size 100 proofed to be to small as we saw an increase in accuracy between 5% and 10% when we increased the vocabulary size to 500.

3. *Apply vector quantisation, which takes the descriptors and maps it to the index of the nearest code-vector, and standardise.* This creates the bag-of-feature representations consisting of a vector of size 500 for each of the training images.

4. *Train the classifier on the bag-of-feature representations.* For the two SVM models we conducted a grid search with 3-fold cross-validation for the following sets of hyper-parameter: $C = \{1, 10, 100, 1000, 10000\}$ and $\gamma = \{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1\}$. For the two logistic regression models the parameter C was estimated via a grid search with 3-fold cross-validation, for C in the set $\{0.1, 1, 10, 100, 1000\}$. The selected parameter are shown in Table 1.

The total run-time of the above pipeline was approximately 10 minutes for each model on a 1.3 GHz Intel Core i5 with 4GB of memory.

Table 1: Results on the "much face" validation set

Classifier	ResNet34	SVM		LR	
Feature	-	SURF	SIFT	SURF	SIFT
C	-	10	10	10	10
γ	-	0.0005	0.0005	-	-
Val. Acc	0.9840	0.8435	0.8275	0.8339	0.7827

Table 2: Results on the "much face plus" validation set

Classifier	ResNet34	SVM		LR	
Feature	-	SURF	SIFT	SURF	SIFT
C	-	100	10	10	10
γ	-	0.0001	0.0005	-	-
Val. Acc	0.9770	0.8362	0.8247	0.7902	0.7701

3.2.3 Inference pipeline

During inference, that is when the "RecogniseFace" function is executed on a test-image, the pipeline illustrated in Figure 14 is executed. In the first stage one or multiple faces are extracted from the image. This stage is followed by either a forward pass through the trained ResNet34 model, or descriptor extraction via SIFT or SURF, followed by vector quantisation using the codebook created in step 2 of the training phase, followed by a pass through the trained SVM or logistic regression model. The final report matrix of size $N \times X \times Y$ contains the predicted class ids N and the central-face-regions X and Y that indicate the location of the faces in the test-image. Consequently the function accepts two arguments: *featureType* and *classifierName*. Valid options for *featureTypes* are "SIFT", "SURF" and "None", valid options for *ClassifierName* are "Resnet34", "SVM" and "LR".

3.3 Results

Each of the five models was trained on the "much face" dataset, containing only faces of persons for which individual photos are available, and on the "much face plus" dataset that contains additional faces of nine unknown persons for which no individual photos are available. While the main focus of this section is on the former dataset, the trained models on the latter are used during inference in order to account for the unknown faces in the group photos.

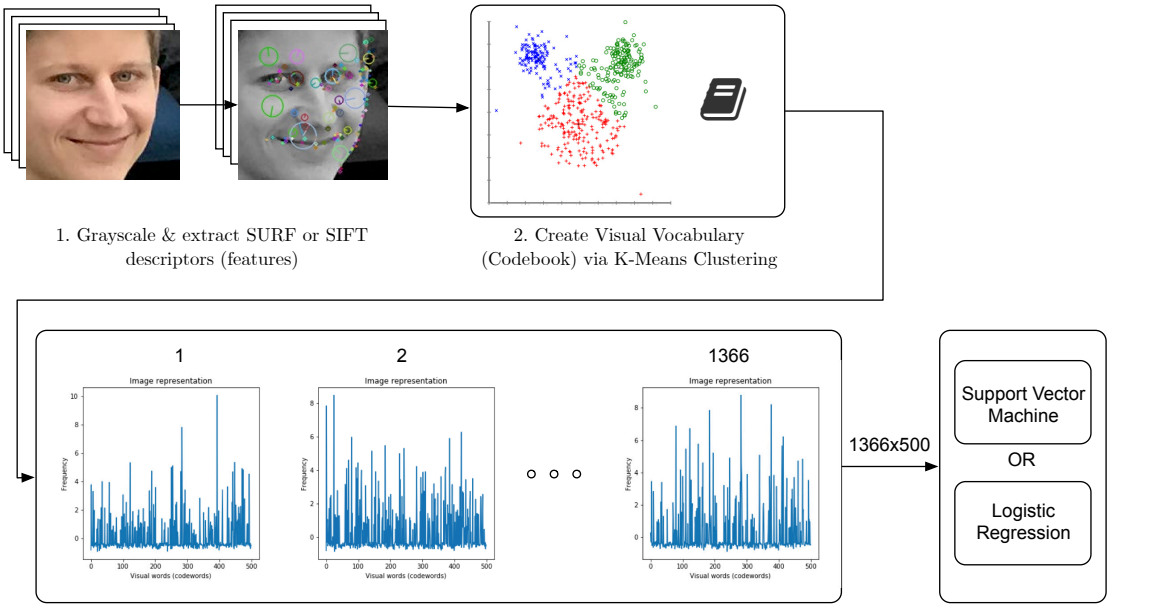

Figure 11: Bag-of-visual-words training pipeline

3.3.1 Results on the "much face" dataset

Table 1 compares the validation-set performance for our five models. The best model, by a wide margin, is ResNet34 with an accuracy of 98.40%. Notably both bag-of-visual-feature models show higher accuracy with SURF, 84.35% and 83.39% for SVM and LR respectively, compared to SIFT (82.75% and 78.27%).

Figure 12 shows the confusion matrix for the ResNet34 model. Out of 313 validation images, 5 images belonging to three classes were incorrectly classified. Figure 13 plots theses along with examples of correctly classified faces for the same individual. Several interesting observations can be made. First, the misclassified image belonging to individual 66 is blurred and could even pose a challenge for a human examiner. Second, while the first missclassified image of individual 69 is slightly distorted, the second one is of good quality. Closer analysis of the training images revealed that these images had very different illumination and facial expression compared to the training images. Last, the two misclassified images of individual 168 suggest that the training images did not contain images without the individual wearing classes. This is not the case, however only two out of 30 training images are without classes, suggesting that our model puts substantial weight on features that help identifying the glasses.

3.3.2 Results on the "much face plus" dataset

All five models perform worse on the larger dataset compared to the smaller dataset as highlighted in Tables 2 and 1. Whereas ResNet34 and SVM model performance degenerated only by around 0.3% to 0.5%, a larger drop for the logistic regression models of up to 4.3 % can be observed. Like for the smaller dataset, models trained with SURF features resulted in higher accuracy compared to the models trained with SIFT features.

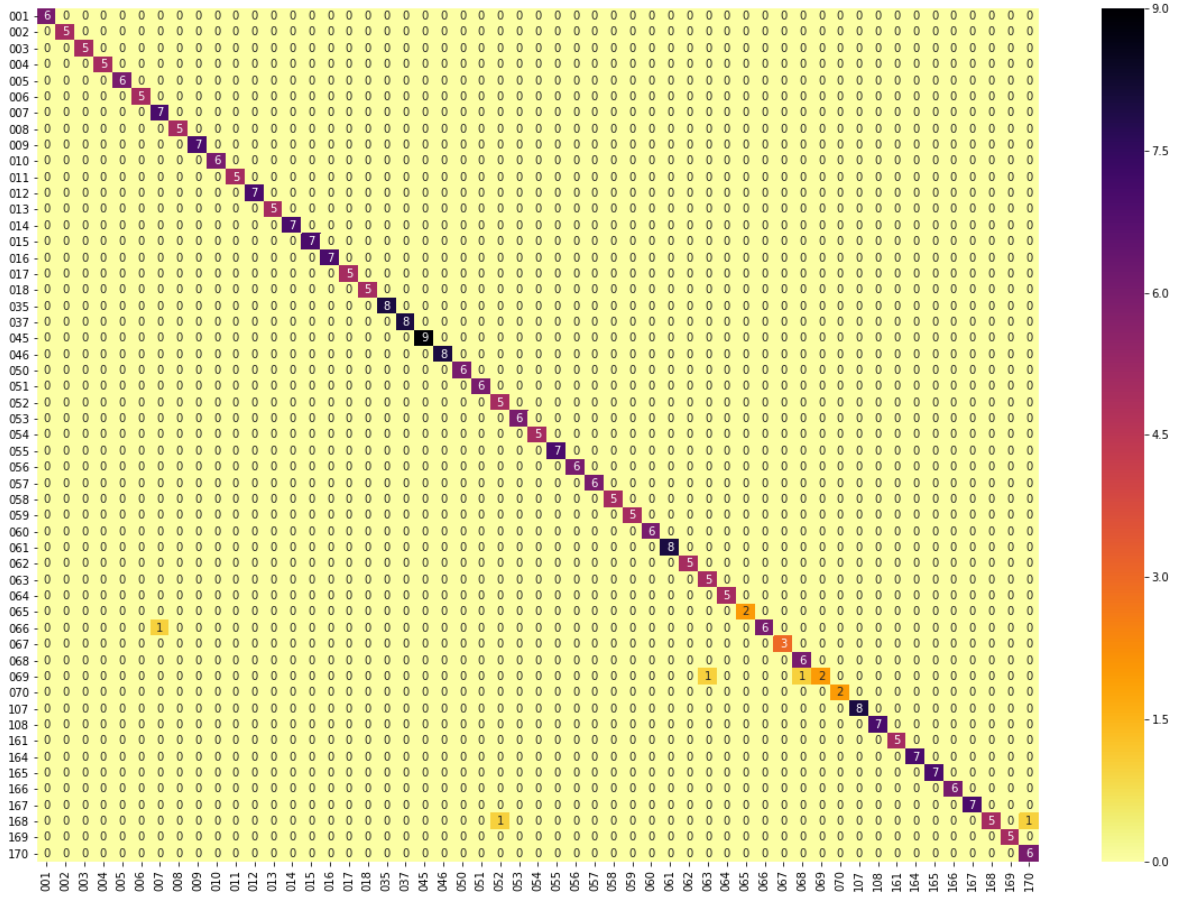

Figure 12: ResNet34 Confusion Matrix

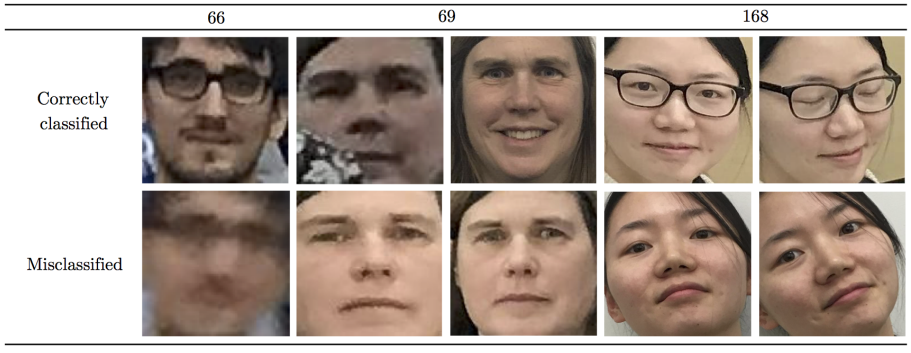

Figure 13: Illustration of the five misclassified faces, along with examples of correctly classified faces

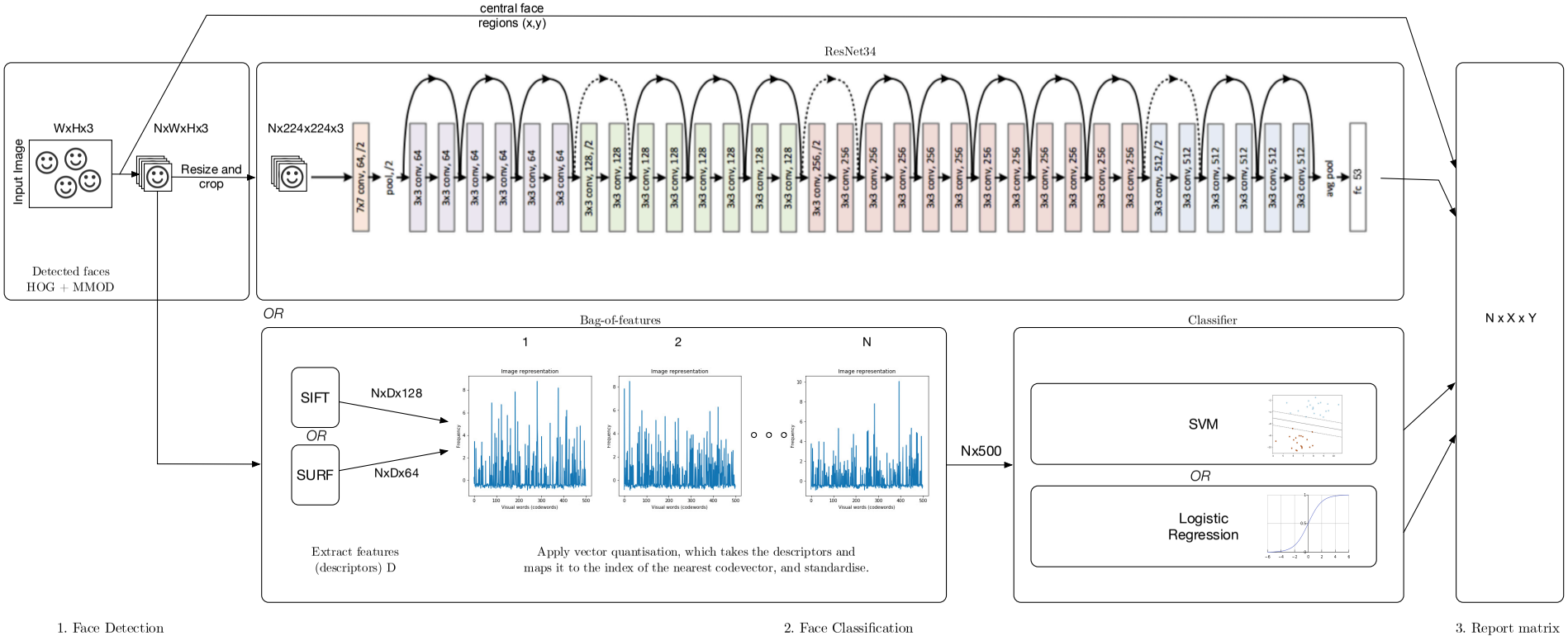

Figure 14: Block-Diagram for the RecogniseFace Function

Figure 15: Thresholding (left) and rectangle detection (right)

4 OCR

4.1 White rectangle detection

The idea and source-code for this part of the OCR pipeline were taken from Rahman K [18]'s post on Stackoverflow. This method consists of four steps:

1. *Threshold the image.* Illustrated on the left-hand side of Figure 15. The threshold value of 200 was determined by manual experimentation.

2. *Find contours via the border following algorithm introduced by Suzuki and Be [19].*

3. *Loop through the found contours*:

    - *Find the convex hull of each contour via the Sklansky algorithm [20]* This step corrects the founded contours for irregularities (convexity defects), in our case the fingers holding the paper.
    - *Approximate the contour shape to a shape with a reduced number of vertices depending on the precision ϵ of the arc length [21].* In our case a good value for ϵ was found to be 0.02.
    - *Only keep contours that have exactly four vertices.*

The right hand side of Figure 15 highlights an example of a correctly identified rectangle.

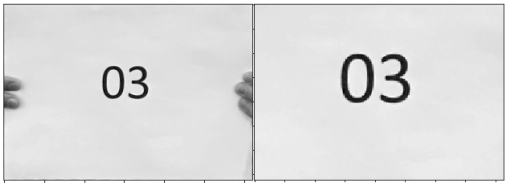

Figure 16: Four-point perspective transformation (left) and centre crop (right).

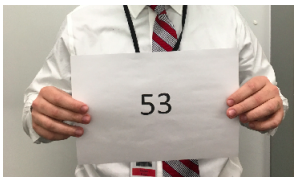

Figure 17: Example of an image with missing black background behind the white A4 paper

4.2 OCR

Before feeding the identified rectangles into the OCR algorithm, two additional pre-processing steps were conducted:

1. *Four-point perspective transformation to correct for differences in rotation and perspective.* The left-hand side of Figure 16 shows the transformed rectangle, which was identified in the image in Figure 15.

2. *Center cropping* to remove the fingers that could result in wrong OCR results as shown on the right-hand side of Figure 16.

In the final ocr step, the pytesseract ocr function [22] is applied to all identified rectangles.

4.3 Caveats

Unfortunately not all images included the additional black background that highlights the A4 paper. A substantial number of images - 262 out of 499, notably those having the ".JPG" extension, were missing this contrast enhancing background, as shown in Figure 17. For those cases our method performed poorly, with an accuracy of only 30% compared to 86.5% for the images that included this background (Table 3). We therefore added a sliding window approach for cases in which the rectangle detection failed. Here we slide a window,

Table 3: DetectNum Accuracy

Classifier	Without sliding window			With sliding window		
Dataset	inc. black background	without black background	combined	inc. black background	without black background	combined
Accuracy	0.8649	0.3091	0.5752	0.9831	0.6946	0.8316

which has the size of 10% of the original image, with a step-size of 64 pixels across the image. On each window the pytesseract function is executed. The discovered text strings are then filtered to include numbers only. The function returns the number with the most occurrence. We acknowledge that this method at it's current implementation is quite inefficient in terms of processing speed, but given the time constraint of the project as a whole it proofed to be one of the easiest to implement.

4.4 Results

Table 3 summarizes the results with sliding-window and without on both sub-sets and the combined dataset. In both cases a clear improvement in accuracy is noticeable. For the combined dataset we achieved an acceptable accuracy of 83.16% up from 57.52%. For images that include the black-background, we even reached an accuracy of 98.31%.

5 Conclusion and future work

In this report we discussed a range of popular techniques used in computer vision and ocr and outlined our implementations of them for face detection and recognition and number detection and recognition. For the former, our best performing method produced an accuracy of 98.40% on validation set. The latter achieved an accuracy of 83.16% on the complete dataset.
While our current methods show good performance, they rely on two distinct stages, face detection and face recognition and rectangle detection and number recognition. In future work we would like to explore an "all-in-one" approach, that combines the two stages into one and is able to solve both tasks at the same time. YOLO [23] and SSD [24] are two example architectures that could be used for such an approach, they require however manual and time-intensive bounding box labeling. To improve the accuracy of our current method, deeper ResNet networks with 101 layer could be tested, additionally image augmentation such as rotation and flipping might further improve performance. In regards to the A4 rectangle detection we would like to train a HOG+MMOD detector similar to the one we used for face detection, which could improve the detection of the rectangles that lack the black-background.
In addition to the provision of the source-code and installation instructions for macOS, we also deployed our implementation on an AWS instance to enable quick experimentation and execution of the DetectNum and RecogniseFace function. Further details are outlined in the accompanying setup-guide.

References

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, Jun. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[2] D. E. King, "Max-Margin Object Detection", *ArXiv:1502.00046 [cs]*, Jan. 2015, arXiv: 1502.00046. [Online]. Available: http://arxiv.org/abs/1502.00046.

[3] V. Jain and E. Learned-Miller, "FDDB: A Benchmark for Face Detection in Unconstrained Settings", University of Massachusetts, Amherst, Tech. Rep. UM-CS-2010-009, 2010.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bot-

tou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[5] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *ArXiv:1409.1556 [cs]*, Sep. 2014, arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556.

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", en, *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec. 2015, ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-015-0816-y. [Online]. Available: https://link.springer.com/article/10.1007/s11263-015-0816-y.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", *ArXiv:1512.03385 [cs]*, Dec. 2015, arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[8] F.-F. Li, J. Johnson, and S. Yeung, "Lecture 9 CNN Architectures", p. 101, May 2017. [Online]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf.

[9] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", en, *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004, ISSN: 0920-5691, 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94. [Online]. Available: https://0.link.springer.com/article/10.1023/B:VISI.0000029664.99615.94.

[10] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-Up Robust Features (SURF)", *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008, ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2007.09.014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314207001555.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, ISBN: 978-0-387-31073-2.

[12] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines", *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, 27:1–27:27, 2011.

[13] *Generalized Linear Models - scikit-learn 0.19.1 documentation*. [Online]. Available: http://scikit-learn.org/stable/modules/linear_model.html#id23.

[14] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments", University of Massachusetts, Amherst, Tech. Rep. 07-49, Oct. 2007.

[15] D. E. King, "Dlib-ml: A Machine Learning Toolkit", *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1755–1758, 2009, ISSN: ISSN 1533-7928. [Online]. Available: http://www.jmlr.org/papers/v10/king09a.html.

[16] D. King, *Dlib 18.6 released: Make your own object detector!* [Online]. Available: http://blog.dlib.net/2014/02/dlib-186-released-make-your-own-object.html.

[17] *Paperspace - Your entire computer in the cloud*, en. [Online]. Available: https://www.paperspace.com/pricing.

[18] A. Rahman K, *Objective c - OpenCV C++/Obj-C: Advanced square detection - Stack Overflow*, May 2012. [Online]. Available: https://stackoverflow.com/questions/10533233/opencv-c-obj-c-advanced-square-detection/10535892.

[19] S. Suzuki and K. Be, "Topological structural analysis of digitized binary images by border following", *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985, ISSN: 0734-189X. DOI: https://doi.org/10.1016/0734-189X(85)90016-7. [Online]. Available: http://www.

sciencedirect.com/science/article/pii/0734189X85900167.

[20] J. Sklansky, "Finding the Convex Hull of a Simple Polygon", *Pattern Recogn. Lett.*, vol. 1, no. 2, pp. 79–83, Dec. 1982, ISSN: 0167-8655. DOI: 10.1016/0167-8655(82)90016-2. [Online]. Available: http://dx.doi.org/10.1016/0167-8655(82)90016-2.

[21] Douglas David H. and Peucker Thomas K., "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature", *Classics in Cartography*, Wiley Online Books, Mar. 2011, ISSN: 9780470669488. DOI: 10.1002/9780470669488.ch2. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/9780470669488.ch2.

[22] M. A. Lee, *Pytesseract: A Python wrapper for Google Tesseract*, original-date: 2010-10-27T23:02:49Z, Mar. 2018. [Online]. Available: https://github.com/madmaze/pytesseract.

[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once-Unified, Real-Time Object Detection", *ArXiv:1506.02640 [cs]*, Jun. 2015, arXiv: 1506.02640. [Online]. Available: http://arxiv.org/abs/1506.02640.

[24] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD Single Shot MultiBox Detector", *ArXiv:1512.02325 [cs]*, vol. 9905, pp. 21–37, Dec. 2016, arXiv: 1512.02325. DOI: 10.1007/978-3-319-46448-0_2. [Online]. Available: http://arxiv.org/abs/1512.02325.