Study on Facial Recognition and Natural Images Text Detection using Matlab

Flavio R. de A. F. Mello

April 26, 2019

1 Domain and Task

2 Preprocessing

Data preprocessing plays an important part in any machine learning based task. In computer vision related ones this is specially true given the often unstructured nature of both the the data collection method - often relying on images scrapped from the web or pictures/videos taken with little or no relevant metadata saved along the media files. For the task at hand, a comprehensive preprocessing pipeline was devised with the aim of gradually transforming the raw dataset provided into a fully labeled dataset primed for being fed into the multiple classifiers tested. As the ultimate objective of this study is of the didactic nature, I opted to ensure each step of the pipeline outputted its results in a new folder. The increased storage requirements posed by the data duplication this choice entails is a reasonable compromise for the ability to inspect and process each step of the pipeline individually, thus facilitating a better understanding of the steps involved in the process and tightening the feedback loop between writing code and perusing its results in the data. In a large scale computer vision task setting, it is reasonable to expect this strategy to be inviable as the storage requirements costs would prove to be too high. Throughout the entirety of the preprocessing pipeline, a mixture of manual, semi-automated and fully-automated solutions are used with the focus on always using the most automated solution possible, while maintaining the quality expected for said step. As will be shown in section **??** there are cases where there was need to fall back to a semi-automated solution due to the poor performance presented by a fully automated one.

Figure **??** shows the complete pipeline implemented for the facial recognition task, including the preprocessing steps - which represent the majority of the effort expended in this study - as well as a legend indicating the level of automation applied to each step.

2.1 File Format Normalization

A common issue when dealing with data science tasks is data formatting. Data is often stored in multiple different formats and schemas, regularly creating the need for normalizing the input before proceeding with the analysis. Computer vision is not free of such trappings. Considering just some of the more ostensibly used formats, there are at least 5 different image extensions (*jpg, bmp, png, tiff, and gif*) and 5 video extensions (*mp4, avi, mov, wmv, flv*). These numbers are increased when taking into account lesser known, and sometimes not open, formats. Considering different tools provide different level of support for each format, it is of interest to tackle this issue early on in the analysis pipeline, ensuring data flows seamlessly throughout the process. Luckily, given how pervasive this format plurality is, there is no shortage of tools available that can convert between the different file extensions.

2.1.1 Image Files Conversion

For the task at hand, there are 2 different image formats present in the data: *jpg* and *heic*. *Jpg*, or *jpeg*, is a longstanding open format with ample support in most ecosystems, while *heic*, or *heif*, is a newer format

proposed as an alternative to *jpg* being able to achieve higher compression. Most notably, it became the standard photo format on iOS 11. Given that the platform being used for the analysis (Matlab) provides support for *jpg*, but not for *heic*, it was decided to convert all the *heic* files in the dataset provided into *jpg*. This was done using the open source tool ImageMagick, which provides support for display and manipulation for multiple image formats. Additionally, the tool is available in all major ecosystems both desktop (Windows, OSX and Linux) and mobile (iOS and Android). Using windows, this can be done with a single line in the command prompt:

```
mogrify -format jpg *.heic
```

This command converts all *heic* images found in the current path into *jpg*, and outputs them in the same folder, while keeping the filename prefix the same.

2.1.2 Video Files Conversion

For this particular task, no video conversion was needed, the video formats encountered were *.mp4* and *.mov*, both fully supported by Matlab.

2.2 Video Frame Extraction

The face recognition techniques used in this study rely on static images only, therefore in order harness the benefits of having video data available, it was decided to extract individual frames from the video files, and include these frames in the pipeline along with the pictures provided in the dataset. This can be done rather efficiently in Matlab as it provides native support for reading individual frames from video content. When extracting frames, it was decided not to keep every single frame, as this would exponentially increase the number of images/faces to be labeled at further stages in the preprocessing pipeline - even though movies taken from individual people (and its resulting frames) are easily labeled, there is still the matter of group videos, where there is no easy way of labeling extracted faces without relying on manual input in some way - this is further discussed in section **??**. Considering this study is for didactic purposes, it was decided to keep only every 5th frame from the videos provided, as a way to balance the time spent in other stages of the facial recognition pipeline while still understanding of the difficulties involved in the data grooming necessary to have an effective pipeline. An additional complication in video frame extraction is that some of the provided videos have a fadeout effect which gradually decreases the image brightness to 0. Although having examples of the same person in different lighting and brightness levels may be favorable to training a model which is more robust against changes in lighting, this may be problematic because some of the frames are effectively too dark for human eye to distinguish the subject, although it is expected for the face extraction to ignore said frames since no faces would be found, in my initial tests, the face extraction step did recognize as faces some undesirable sections, thus requiring further manual input to remove these pitch black images from the labeled dataset. As an attempt to diminish this effect, when extracting video frames, a minimum brightness threshold is used to ignore images that are deemed as "too dark". It is important to note how the addition of video based data, enabled by the increased dissemination of portable electronics able to record high definition videos, is an important upgrade on static pictures based data only. Videos capture slight changes and nuances in the image which prove valuable when training an image based classifier. Considering that the effort, time and cost required to record a video is nearly the same as a picture, and that videos provide orders of magnitude more frames to be processed, it would be sensible to resort to video based data unless the increased definition provided by static pictures is required.

2.3 Face Extraction

Having already preprocessed all videos and labeled the individual content, the next step in the preprocessing pipeline is to extract individual faces from the entire content (i.e. both single and group images/frames). This can be done in multiple different ways and, in part, also depends on how the latter stages of the facial recognition pipelines are designed. For example, if the recognition techniques rely on a specific size or aspect ratio, it may be best to include these constraints when extracting/cropping the face regions instead of simply

resizing images at a latter stage as, such resizing can introduce artifacts or distortions that may decrease the accuracy of the trained models - this is specially true for changes in the aspect ratio. In this study, the face extraction stage can be divided into three substages: face detection, face cropping, and data cleaning.

2.4 Face Detection

In order to extract faces from a natural image, one must first locate where the faces are localized. In this study, this was done using Matlab's pretrained 'FrontalFaceCART' Cascade Object Detector which relies on Haar features to locate faces within an image.

2.5 Face Cropping

Once each face region is detected they are then cropped, resulting in a new image containing only the face. The method of doing may depend on how latter stages of the pipeline are setup, as mentioned above. In this study, it was decided to work with square images of faces - based on University of Oxford's Visual Geometry Group approach, taken in VGG-Faces. That is, when cropping the facial regions, ensure the cropped region is a square one, and already resizing the resulting crop into the exact dimensions used to feed images to the CNN. Even though some of the feature types used for non-CNN classification techniques are scale independent (e.g. SURF), it was decided to include the resizing step in all faces for the sake of simplicity and keeping the preprocessing pipeline unified. Therefore, the resulting image of face cropping is a 227x227, 3 channel RGB image that was resized using bicubic interpolation, in order to conform with alexnet's requirements.

2.6 Data Cleaning

It is expected for the face extraction algorithm to present some false positives (i.e. regions that are not in fact faces). It is desirable then to remove these images from the labeled sets so as not to affect the classifiers performance - having sections of peoples clothes could artificially increase its performance as all pictures were taken in the same day, with people keeping the same clothes in all pictures; conversely, having sections of the background along with the labeled faces could decrease model's accuracies as pictures were taken in the same locations, making the same background features visible in pictures from different subjects. For the purposes of this study, cleaning is done by manually going over all extracted sections and selecting ones which are not in fact faces. Finally, this collection of non-faces is saved for further use - this will be discussed in section **??**.

2.7 Face Labeling

Even though techniques such as CNNs rely on unsupervised learning concepts to determine the features to be extracted, facial recognition ultimately relies on supervised learning to match features to labels - in this case, numbers assigned to every participant. In this implementation, such labeling is done at two different stages in the pipeline. The first one, done before starting the preprocessing steps of image format conversion, the individual pictures and movies were already separated into individual folders with names matching each one's assigned label. Although this was done manually, it was done somewhat efficiently in the dataset provided, as each individual's pictures were taken sequentially and, thus images' metadata such as filename and creation date can be used to easily sort images in a way such that all images from a given person are found in a contiguous group. The second labeling task refers to labeling faces extracted from images/videos of the whole group. These provide a much more difficult task since there is no immediate way of bundling faces. Considering that a) there are 69 different individuals (i.e. labels) b) their respective labels are assigned arbitrarily and c) the human short-term memory capacity is restricted to the magnitude of 7 different items; any fully manual strategy of sorting the pictures will surely take too long and consume valuable time and resources. If we are to consider the applications of this pipeline in similar problem sets with a larger amount of labels, the resources needed to manually label each individual face would increase exponentially. As an

effort to diminish the time taken to label these faces, I took the semi-automation approach. That is, to combine automatic and manual techniques so that the manual portions are enhanced, and therefore sped up, by the automation. For this purpose, two separate automation approaches were taken: I) Classification and II) Clustering. Both strategies and their perceived gains and pitfalls are stated below in sections **??** and **??**.

2.7.1 Label Automation: Classifier

As stated above, we already have at our disposal a subset of the face images that can be easily labeled (the ones taken from individual pictures/videos). Therefore we can train a facial recognizer with this subset only, and use it to predict the labels for faces taken from group data. This classifier will still perform worse when compared to one trained with the full dataset, especially considering the group pictures are the ones which introduce much of the scaling problem (i.e. faces from people in the back of the group have considerably smaller resolutions than ones from people in the front plane. Individual pictures were all taken at similar distances and resolutions). However, it still can be used to augment the labeling process. As limited as human short-term memory is, the human brain still excels at image comparing tasks, that is: to tell if two or more images are from the same subject. By leveraging a facial recognition classifier at this stage of the preprocessing pipeline we have the double gain of making the process more efficient, as well as creating a loose benchmark to compare how the classifier performed when trained with individual pictures only versus when trained with the full dataset, thus being able to gauge the effect of introducing the group subset in the train data. Implementation-wise, the strategy chosen was an iterative one. That is, establish a minimum prediction confidence threshold and label only predictions made with confidence level above said threshold. The intuition is that, by running this cycle multiple times (training and labeling), trained models' accuracies increase as more labeled data is made available which, in turn, enables the next trained model to confidently label part of the remaining data. In practice, however, this approach was not successful. The model tested was a Random Forest of 700 trees trained with the 500 most relevant SURF features. Based on the individual pictures alone, the trained model was not able to accurately classify faces extracted from the group shots. Even when restricting to the classifications made with the highest confidence (30-50%) the classifications were often wrong. Based on these results, I opted to abandon this approach.

2.7.2 Label Automation: Clustering

After the failure of the classifier labeling approach, I decided to follow a slightly less automated one: clustering. In this methodology, instead of trying to predict the label of each face individually, all the faces extracted from the group data (over 30 000) are clustered using K-Means and images are then exported to separate folders according to their cluster id. This method does not have the benefit of automated labeling in the sense that it still requires manual input in the form of assigning a label to each face by recognizing the face and moving the file into its respective label folder. However, by grouping like images in folders some presorting is effectively made. Now it is more probable that a series of images belonging to the same label are ordered next to each other, furthermore, each folder contains only a small subset of different labels - in this experiment it was not rare to have folders with images from a single label and most folders containing up to 5 different labels. This setup vastly improves the labeling speed, by traversing a list of like images, the human vision system is able to quickly identify if any outlier is present. This method relies, then, on having as many images from the same person in a row as possible so that the label discovering is done only once for a large group of images. Needless to say, the number of clusters used plays an important role in providing proper grouping. Smaller amount of clusters increases the likelihood of having individual clusters with multiple labels. Conversely, if the cluster number if too large, what would be a single large chunk can be separated into multiple clusters. For the purpose of this task, I opted to cluster the data into 100 clusters (approximately 1.5x the amount of labels) to provide some separation while keeping the number of folders manageable. I then proceeded to labeling the data iteratively by focusing on these larger chunks of images, and ignoring groups/folder which had too may different labels to enable a efficient form of selecting large groups. Figure **??** shows a few examples of the data resulting from clustering.

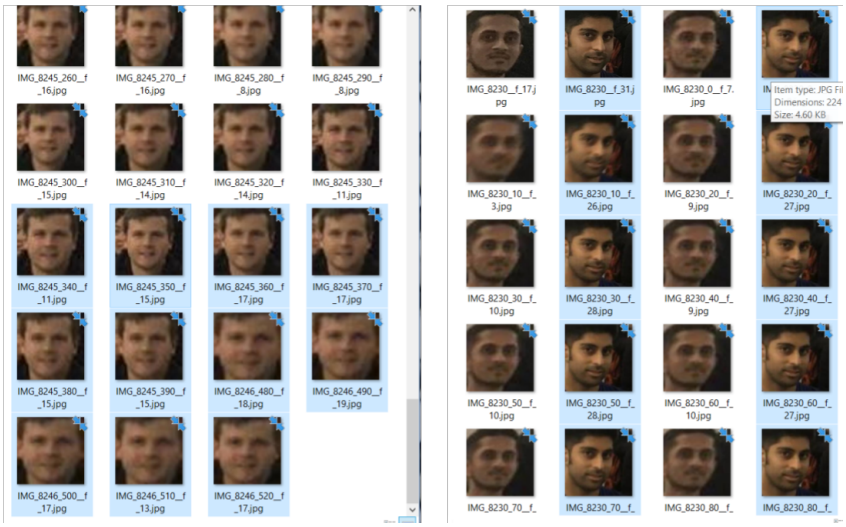

Figure 1: Examples of image groupings resulting from clustering. Contiguous groups (left) are preferred, but some patterns allow for quick vertical selection as well (right)

Once all the larger groups were labeled, I simply reclustered the remaining data into another 100 clusters, as the vast decrease in unlabeled data would naturally cause new clusters to be better grouped. Table **??** shows the amount of labeled images per clustering event. After only 3 epochs, the complete dataset of faces extracted from group pictures/movies was processed.

Cluster Epoch	Sorted Images	Acc. Sorted %
1	23527	65%
2	7587	86%
3	2410	92%

Table 1: Group Image Labeling Through Clustering Progress

For the improving the efficiency of label recognition, I set up a simple visual reference chart by selecting one sample image from each individual and naming them according to their respective labels. This provided a fast and important visual cue for remembering every single label. Figure **??** shows this reference sheet.

2.7.3 Label Results: Outliers and Unlabeled

During the labeling process a few outliers were noted they are present in this section:

2.7.4 Unlabeled Individuals

In the group data, there are some individuals who are not present in the labeled individual pictures. This could be either because they simply not present in the lab section when the labeling was made, or they chose not to be labeled. In this particular case, I opted to ignore these images and not include them in the dataset. Despite knowing the face extraction step will still select these faces when presented with a group image, and they will inevitably be classified as another individual - one of the properly labeled ones. This choice was made based on the fact the, prior to training the classifiers, the full dataset is trimmed to ensure all classes have the same amount of data to ensure there is no distribution bias in the models: That is that they classify based on the features alone, and not on some expected distribution of the classes. Each face has the same probability of appearing, and any discrepancies in the amount of data available is either due to minor changes in the human data collection (e.g. videos of not the exact same length) or, in other cases, or due to difficulties in the face detection algorithm used (further explained in section **??**). If I was to include these

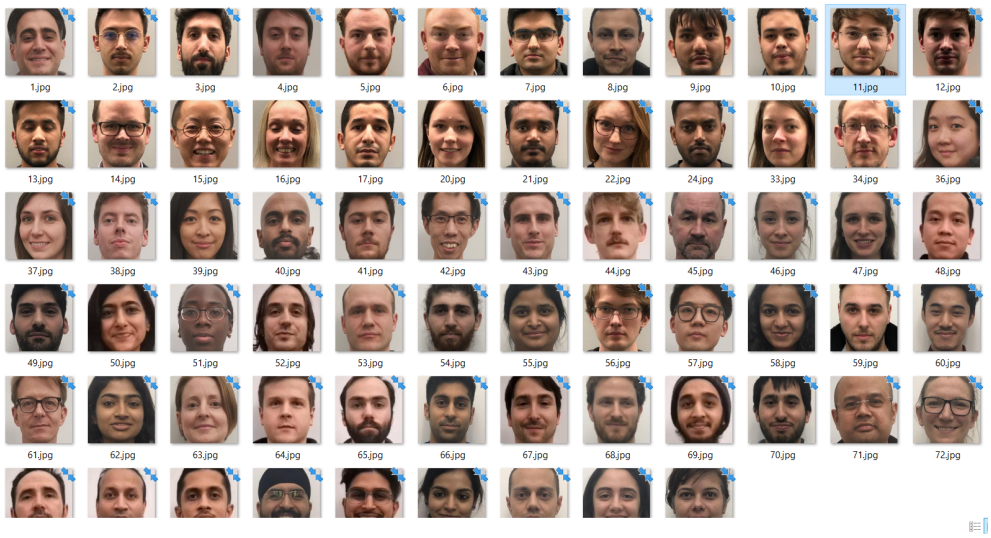

Figure 2: Labeling Reference Sheet

faces as labels, the amount of data used for training would be vastly reduce given that these individuals have considerably less examples than the other ones. Figure **??** shows the individuals in question.

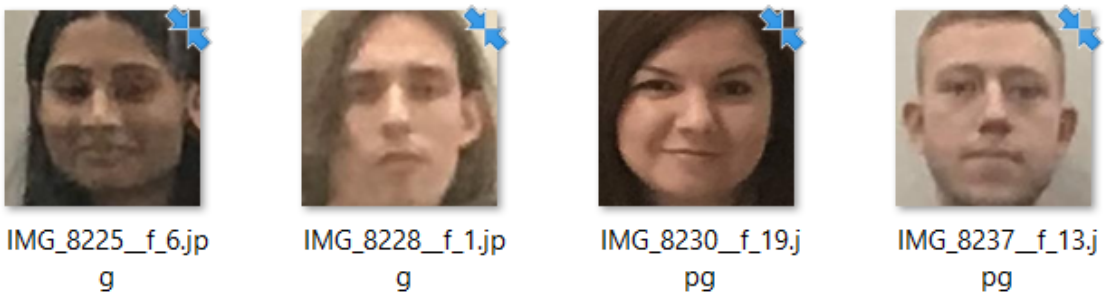

Figure 3: Individuals that did not have individual pictures with labels

2.7.5 Unlabelable Images

Considering the faces were also extracted from video material, which is of considerably lower resolution than the images, some of the faces were too unfocused/blurred for me to be able to confidently label them. As they are singnificantly small in number when compared to the rest of the data, I opted to remove them from the dataset instead of risking introducing erroneous data by human error. Figure **??** shows some examples of images that were left out of the final dataset.

2.7.6 Face Extraction Difficulties

The face detection algorithm used leverages the Viola-Jones algorithm with Haar features to detect changes in shade in specific regions of the face, such as eyes, nose and forehead. It is known that this particular type of detection strategy is inherently biased (see http://gendershades.org/), being able to detect lighter skinned faces with considerably better than darker skinned ones. This effect was observed in more evidently in the faces extracted from the group data where one individual had around 20% instances of her face detected when compared to the rest of the data. It is of the utmost importance to stress how this type seemingly innocent of bias could lead to unwanted biases in other critical applications of computer vision and should be combated, specially considering that face detection often lies at the initial stages of a computer vision based pipeline.

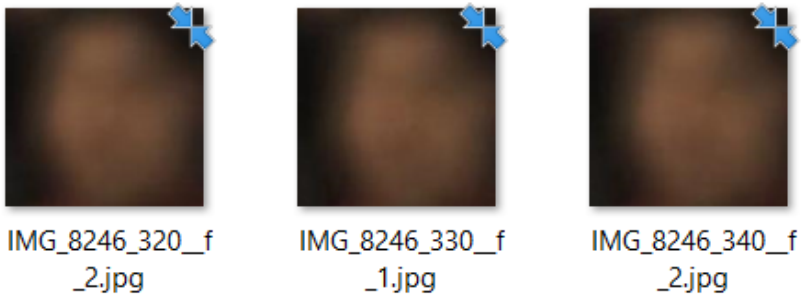

Figure 4: Images that were left out of the dataset due to extreme poor quality

3 Facial Recognition

Once all the preprocessing and labelling is done, the actual image classification task is somewhat simple as there are powerful tools readily available for the task not only in Matlab, but also in other languages/stacks as well. This study focused on 2 visual feature extracting techniques - SURF and HOG - as well as 5 different classification methods - Convolutional Neural Networks (CNN), Naïve Bayes (NB), K-Nearest Neighbours (KNN), Support Vector Machines (SVM), and Random Forests (RF). The four latter methods are classical machine learning classification methods, which were adapted for image classification based on natural language processing techniques such as bag of words - treating the occurrence rate of each individual word in a document as a separate feature. For image classification, "visual" words are used, that is a collection of visual features is extracted from each image, and the occurrence rate of each feature (word) is the feature vector. Both SVMs and RFs then rely on using a graphical feature extracting technique (SURF or HOG in this case) to generate a feature vector based on the occurrence rate of said features in each image, and then feeding said feature vectors, along with their respective labels, to the classifier in order to train it. Predicting a label then requires the user to extract from the new image the occurrence rate of the same visual features, and feeding this into the classifier. CNNs, on the other hand, are different from classical machine learning approaches in that the neural network itself performs both the feature extracting and classification - this is further discussed in section **??** below.

3.1 Histogram of Oriented Gradients - HOG

Histogram of Oriented Gradients is a somewhat simple feature descriptor that relies on sectioning an image in to a grid (with some overlap) and computing the orientation gradient for each of the local sections. When used for image classification, each section's gradient becomes a single feature, it requires then for all images to be of the same size so that the grid sections are comparable. It is, therefore, not particularly robust against changes in scale or orientation.

3.2 Speeded-Up Robust Features - SURF

Based on scale-invariant feature transform (SIFT), the Speeded-Up Robust Features (SURF) is another visual feature descriptor which is able to extract features in the form of image regions (i.e. blobs) that are invariant to scale and rotation. That is, a single feature represents the same visual blob at multiple different scales and rotations. This is achieved by applying filters of multiple different sizes. This is made possible because SURF relies on the use of integral images, which makes the computational cost of applying a filter constant, regardless of the size of the filter.

3.3 Support Vector Machine - SVM

Support Vector Machines are a family of machine learning algorithms often used in classification tasks. Even when resorting to a linear kernel, they are able to separate non linearly separable data by mapping the data

to a higher dimensional space, one in which data may be linearly separable.

3.4 Random Forest - RF

Random forest is another instance of machine learning algorithm that can be used for classification or regression tasks. Its name derives from the fact a random forest relies on growing multiple decision trees based on the data provided. This is done with feature bagging (i.e. each tree is grown with a random subset of the features) to reduce the bias of the forest as a whole. Once all the trees are grown (i.e. the forest is model trained), prediction is done by traversing all trees with the new sample data, and voting between the trees resulting classification.

3.5 Naïve Bayes - NB

Naïve Bayes is a family of classifiers that rely on Bayesian inference. Prior probabilities are computed from the training data and used when prediction values for new observations. It presumes that the features are independent between themselves which is often not true in complex scenarios.

3.6 K-Nearest Neighbours - KNN

K-Nearest Neighbours is a simple yet powerful classification algorithm that predicts labels based on the euclidean distance from the new observation to the training data, assigning the label according to the majority of the K-Nearest datapoints found in the training data.

3.7 Convolutional Neural Network - CNN

Convolutional Neural Networks are a newer class of learning algorithms based on deep neural networks. They are considered "end-to-end" visual learning algorithms in the sense that a single CNN provides both the feature extraction and machine learning (most often than not, classification) capabilities. Feature extraction is done by use of multiple convolutional and pooling layers which have the effect of consolidating increasingly large receptive fields into a smaller number of values. Furthermore, the exact filters applied in the convolutional steps are randomly initialized. It is through learning, made via the backpropagation algorithm, that the most effective filters - and, therefore, visual features - are gradually selected within the network. Given the the sheer computational power required to run the backpropagation algorithm through a deep neural network, CNNs were only made possible by the exponential increase in computational power that happened over the course of the last 2 decades. The advent of discrete graphical processing units (GPUs), which excel at highly parallelizable tasks, were also an important factor in making CNNs feasible.

3.8 Unrecognized Faces

As exposed in section **??**, it is more than expected for the facial face detection algorithm to have some false positive rate. This is true at the prediction stage just as much as it is during the preprocessing portion of the pipeline. However, at the prediction stage it is not possible to leverage human input to ignore these regions. One solution for dealing with this problem is to collect all the false positives found in the training stage and treat them as belonging to a new label representing all instances of non-faces extracted from the provided data. Then, if a given face is predicted as belonging to this label, it is ignored when outputting the results of the RecognizeFace function. This was the strategy implemented in this study. An alternative, which was not pursued, could be to ensure the predictors output, along with the predicted label, a confidence score and establish a minimum threshold, ignoring any predictions with scores below said mark.

3.9 Training Process

3.9.1 Data Partitioning

As mentioned in section **??**, there are some discrepancies in the amount of data available for each individual. To ensure no bias is introduced in terms of data distribution, I opted to trim the dataset, forcing all classes to equally distributed in the dataset used. After said trimming process, the resulting balanced data is split into train and test sets according to a 85/15 ratio while keeping both subsets balanced as well. In both the trimming and train/test split, data is selected randomly from each label pool to be a part of the new dataset by use of the `'randomized'` flag in the `splitEachLabel` function.

3.9.2 Conventional Classifiers

For the conventional classifiers used (NB, KNN, SVM and RF), the training process is broken down into 2 steps. Feature extraction and training. Feature extraction entails in using either feature descriptor (SURF or HOG) to transform the raw images into a bag of features, which is then ready to be fed into the classifiers along with the image labels.

HOG

HOG features are extracted according to an user-defined cell size which is used to traverse the complete image (with some overlapping) and calculate the oriented gradients for each subsection of the image. The number of features generated when using HOG is, therefore, a function of a) the image size, b) the cell size and c) the overlapping size. It is important for classification tasks then to ensure the exact same values for these three parameters are the same for all observations (training, testing and new predictions). For this experiment HOG features were extracted using an 8x8 cell, images were all 227x227x3 (RGB channel) and overlapping was kept at 50%.

SURF

SURF features are different than HOG ones in the sense that the number of features present in an image is variable and depends on the image contents as well rather than only in metadata and hyperparameters. It is necessary then for some form of feature selection to ensure all observations have the same amount of features - this is done by way of a feature bag. The original SURF features are extracted from every single image in the training set, the weakest ones are dropped (20% in this study) and the resulting visual features are then clustered, using K-Means, as a form of quantization. These N clusters then become the bag of features which is used to encode raw images into N features. In this experiment N was kept at the default value - 500. This generate bag of features must be saved for future use in order to be able to transform new observations in the form of raw images into the feature vectors used in the classifiers. One important point is that, in this particular experiment, I was not able to build the bag of features using the complete dataset due to resource limitations on my development machine. The bag creation process is memory intensive, when using the complete dataset, it easily depleted the available memory (16GB) and even used some disk space as swap memory. Even then, the command failed in Matlab due to out of memory errors. The solution then was to build the bag of words using a subset of the data (50% selected randomly while keeping class balance, totaling 8085 images), and then encode the complete training data before feeding it into the classifiers. Given the restrictions imposed by the hardware, this was a sensible compromise: even though the bag was built with a subset of the data, the high sampling rate from the video material means that consecutive extracted faces are highly similar and, therefore their underlying visual features should be as well. Furthermore the full training/testing sets were subsequently used to train and evaluate the models.

3.10 Prediction Process

The prediction process is somewhat straightforward. Given an image, the program leverages the same functions used in the preprocessing pipeline to detect, crop and scale the faces, resulting in two arrays: One containing the 227x227x3 images of the extracted faces, and another containing the X and Y points indicating the centre of the facial respective facial regions detected. Centre is calculated as the centrepoint of the rectangular bounding box detected by the cascading face detector. Then the program determines which model should be used based on the `featureType` and classifierName variables provided by the user; loads said model; extracts the corresponding features from the faces array (if necessary) and proceeds to predict the labels using the selected model. Once the labels are predicted, a filtering step is applied to remove from the list faces that were predicted to belong to the *non_faces* artificial label. Finally, the response matrix is assembled to conform to the specified format. Note that, for the feature extraction step, it is necessary to reuse the same featureBag used for training the SURF-based classifiers and the same cell size used for the HOG-based ones.

3.11 Quantitative Results

The results, shown in table **??**, indicate some clear differences between the feature types and classifiers used. In terms of accuracy, the classifiers performed quite well with most classifiers staying in the 75%-85% accuracy range. The best classifier in terms of accuracy was the CNN with 90% correct predictions in the test set, but that came at the expense of a considerably longer training time, over 2.5 hours in total.

The CNN was trained using transfer learning from the pretrained AlexNet, a general purpose object categorization CNN trained with the ImageNet database to recognize over 1000 object categories. It is interesting to note how a CNN trained on a different domain (object instead of face classification) still fared remarkably well when applying simple transfer learning. If the CNN was to be trained from scratch, it would be reasonable to expect better performance given the features captured in the initial convolutions/pooling layers of the CNN would be more relevant to the task at hand. On the other hand, we could also expect the total training time to be orders of magnitude longer. Comparing HOG and SURF based classifiers, the SURF-based ones were one order of magnitude smaller in size and faster to train. The training time becomes less discrepant if we factor in the time necessary to build the feature bag (SURF only) extract the training features, as SURF required ????s while HOG required only ???s for this preprocessing task. Despite being more efficient, the SURF models were, in general, less accurate than the HOG-based ones. It is important to note, however, that one of SURF's most relevant improvements over HOG - being invariant to scale and rotation - has diminished effect in this specific problem set because a) all faces are in the general upright direction and b) the cropping+scaling done in preprocessing means that the faces all have similar sizes when

Feature Type	Classifier	Training Time (s)	Prediction Time (s)	Test Set Accuracy	Model Size (GB)	Obs.
HOG	NB	800	58	78%	3.6	-
HOG	KNN	71	340	87%	3.4	5 NN
HOG	SVM	6030	394	88%	4.4	Linear kernel
HOG	RF	612	28	84%	4.2	100 Trees
SURF	NB	87	525	65%	0.34	-
SURF	KNN	0.57	5	80%	0.033	5 NN
SURF	SVM	61	9	75%	0.056	Linear kernel
SURF	RF	74	2	84%	0.68	100 Trees
CNN	CNN	9194	10	90%	0.23	Transfer Learning based on AlexNet

Table 2: Facial Recognition Results by Feature and Classifier Type

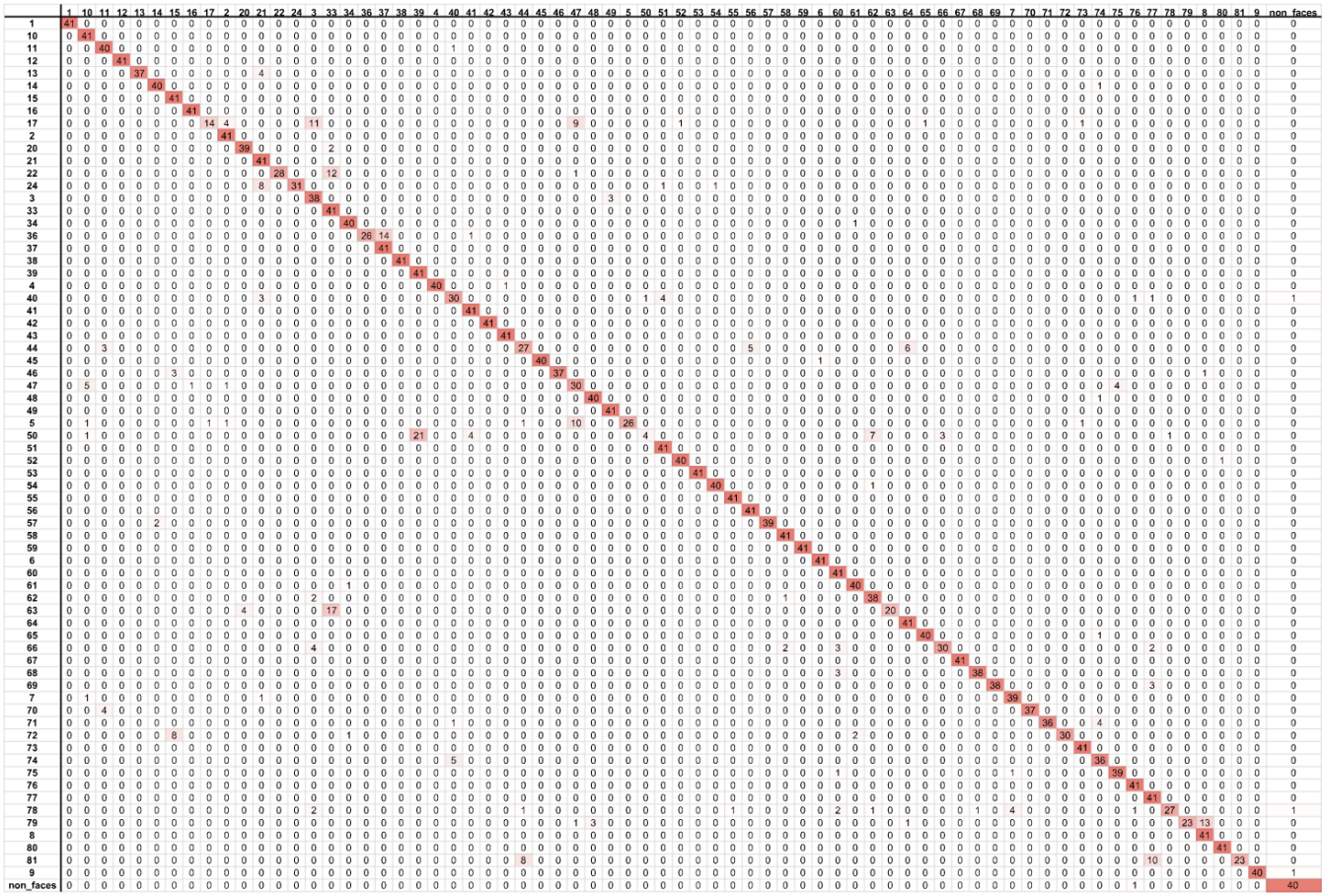

Figure 5: Confusion Matrix of test set predictions by the CNN classifier

being fed into the classifiers. Finally, a confusion matrix for the most accurate classifier (CNN) can be seen in image **??**, the matrix was color coded according to an sequential scale easier highlight misclassifications.

3.12 Qualitative Results

Observing the confusion matrix shown in image **??** and its most common misclassifications yielded some interesting results. Figure **??** show three pairs ([39 50], [36 37], [8 79]) that ranked among the most commonly misclassified instances. In all three cases, one can infer the reasons why the classifier may failed to properly distinct between the classes: subjects have similar skin tones, hair color and haircuts and their faces are in similar positions (either front facing or at an angle) inside the cropping box.

Testing the classifier on a group photo shows how the addition of the *non_faces* artificial label proved to be useful in refining the classification results. Image **??** shows a sample group picture with red annotations indicating false positive face detections (i.e. non facial regions that were deemed as such by the face detection algorithm) and were correctly classified as non faces and, therefore, removed from the returned results matrix P. Finally, image **??** shows the same group photo, this time annotated with the returned predicted labels for each face found. Along the left edge there is a face (belonging to individual labeled 74) that was not detected by the face detection algorithm (false negative), this could be due to the fact the his face is only partially visible, more importantly, his right eye is partially cropped in the photo - eyes are important anchor features in the face detection algorithm used.

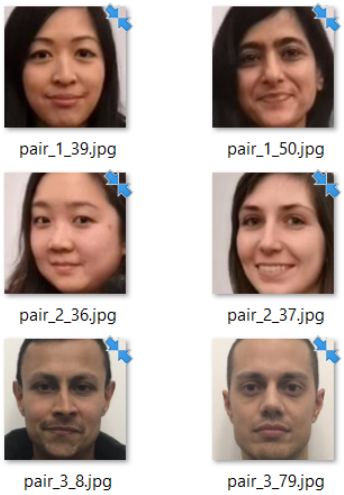

Figure 6: Examples of images misclassified by the CNN classifier

4 Digit Recognition

The second part of the study regards text recognition within natural images. The text in this case was the numbers used for assigning the individuals labels that were used in the facial recognition problem. FOr assigning such labels, the individual pictures and movies were taken while having the person hold a white sheet of paper with the number (i.e. label) printed in black using an unknown typeface. The challenge then lies in recognizing the numbers found in a given picture/video.

4.1 Strategy

One of the first, higher order, problems found is how to deal with different media types - videos and pictures. The solution to this problem chosen was to reduce both media instances into one by treating videos as a series of static pictures. With this strategy, if a solution is found for finding the labels within pictures, finding them in videos becomes trivialized by simply extracting all the individual frames in the videos, feeding them one by one to the label detection algorithm, and deduplicating the results, consolidating them into a unique set of labels found. Once the problem of media duality is solved, the next logical step is then to determine which technique should be used to properly detect and extract the labels from images. For this purpose, I based my strategy on Matlab's tutorial on finding text in natural images. This strategy relies on finding MSER candidate regions in the provided image representing individual characters; filtering out poor candidates using a rule based approach; merging, where applicable, the remaining candidates into contiguous regions and finally performing OCR to find the text in each of these regions.

4.1.1 Filtering Characteristics

Some examples of characteristics used to filter out poor candidates are aspect ratio ($width/height$) and stroke width width. Print letters/digits usually have a well defined interval of possible aspect ratios, too wide sections are disconsidered. Stroke width variation is a somewhat more complex concept, and required further tuning to properly detect numbers in the typeface used in the experiment. Stroke width is calculated by first converting the color/grayscale image into a binary one. The binary image is used to define a skeleton template, that is a fine line that is equidistant to the region's border. Finally, stroke width (and its variation within the region) is calculated by measuring the distance between the skeleton and the region's borders. This is a useful parameter for filtering out candidates because print typefaces often have limited stroke width variation. However, some tuning had to be made regarding this parameter's threshold as the typeface used did have some variation, especially in the curved digits such as 6 or 5. Figure **??** exemplifies this

Figure 7: Group photo: Highlighted in red are non facial regions picked up by face detection and correctly filtered out by the CNN classifier

phenomenon by comparing side by side the same digits in this and the widely used Arial typeface. Notice how, the typeface used in the labels tends to become slimmer in certain regions such as the tip of the 6 or the middle of the curves found in 6 and 5, whereas Arial has the same width across the entire characters. Based on this particularity of the typeface used in the labels, a small change had to be made in the example code to increase the stroke width variation threshold to allowed. This has the effect of increasing the false positive rate (i.e. non text regions that are not filtered out), but this can be further dealt with in the latter stages of the OCR pipeline.

4.1.2 Region Merging

Initial tests with the complete pipeline shown cases in which the individual digits that compound the label were being identified as individual words, instead of single number containing both digits. This proved a valuable hint to indicate that the MSER regions for each digit were not being properly merged into a single text block - one possible explanation could be because the tracking/letter-spacing present in the used typeface is larger than usual. To account for this phenomenon I decided to increase expansion factor applied to merge regions from 0.02 to 0.05. This proved to be a positive change as the correct identification rate increased from ??% to 87%.

4.1.3 OCR

The final stage of the OCR pipeline used in this study is to feed the remaining candidate regions into Matlab's stock OCR function, which allows for various parameters/hints to be set to customize the OCR being applied. Among the available options the two most relevant ones are: `TextLayout`, which can be used to indicate the expected text format being a single line, a single word or a contiguous block of text; and `CharacterSet` which can be used to constrain the character set to a specific collection. Considering the labels are formed by a pair of digits, I opted to use the TextLayout:Word hint, which proved useful in

Figure 8: Group photo: Labels predicted by CNN classifier annotated in red above individuals' faces

eliminating noise returned by the OCR function. The CharacterSet hint, however, proved to have negative results. Restricting the expected character set to the 10 Arabic Numerals effectively introduces bias into the OCR algorithm, which increases the chance of any remaining non-text or textual but not numeric regions to be outputted as numbers. To work around this issue, I opted to take a different approach: instead of constraining the character set to the 10 digits, use only the TextLayout:Word hint and use the knowledge of the problem's constraints to further filter out the text results - keeping texts that are constituted only by numbers and disregarding other textual regions. This proved to have better results - testing against the provided individual data, the final pipeline correctly identified the labels in 87% of the cases. For the purposes of this test, a successful identification is determined if the label number is found among the returned numbers found by the OCR pipeline. There is still the issue of false positives, which remains to be dealt with.

4.1.4 Multiple Numbers

Because of the bottom-up strategy used, the OCR pipeline implemented is able to detect multiple numbers in a single image or video, since each region is independently accepted or filtered out based only on its own characteristics and underlying content.

5 Conclusion

Reflecting on the results obtained by both the label recognizer and face classifier I consider this experiment to be successful. Over the course of the study several different techniques were applied to resolve specific issues which resulted in an overall reasonably accurate pipeline. It is a often repeated mantra that in the A.I. domain, "more data beats a cleverer algorithm ". That is because the current state of algorithmic solutions, especially neural networks based ones, rely on a vast amount of data to be able to properly generalize. On this context, I attribute the success of the facial recognition classifiers trained mostly to the large amount of data I was able to label, especially using the clustering technique described in section **??**. Although it

Figure 9: Stroke width variation sample. Typeface used in experiment (left) vs Arial (right)

still required a reasonably large amount of manual input to label the images, it proved to be a significant improvement over manually labeling the images one by one. Still in the realm of labeling, if the initial labeling of the individual photos were to be more time consuming (e.g. having more labels and/or pictures; files not being grouped by label) a reasonable alternative could be to leverage the digit detection pipeline developed in section **??** to automatically assign labels, in which case, however, the high false digit detection rate would prove to be problematic. A possible solution to that could be to leverage machine learning algorithms to further filter textual region candidates based on their parameter values instead of relying only on a simplistic rules-based approach. Finally, it was interesting to observe the variability in terms training time and model size between the different algorithms and feature descriptors. Regarding future work, a vast horizon remains to be explored. One could study the differences found in models based on different data subsets (e.g. models trained on individual pictures prediction faces extract from group shot. Alternatively, models trained on faces extracted from the group data predicting faces from individual shots), in fact, the entire preprocessing pipeline was structured to allow for this differentiation, keeping the sources separated into subfolders, however, this particular comparison was removed from the final experiment due to time constraints. There is also definitely space for further tuning the hyperparameters of the experiment, both in terms of feature descriptor (e.g. SURF/HOG cell size) as well as classifier model (e.g. number of trees in RF, kernel used in SVM, etc). In the CNN realm, one could look into the differences found by comparing models based on 1) transfer learning from another domain (e.g. AlexNet - general object classification), 2) transfer learning from the same domain (e.g. VGG-Faces - facial recognition trained on a different set of people), and 3) a CNN trained from scratch for this particular purpose. There is also much room for improvement in the preprocessing pipeline, for example rotating the image so that both eyes are leveled in every image cropped. And, finally, one could further explore the analogy of the visual bag of words with other Natural Language Processing (NLP) techniques such as Term Frequency/Inverse Document Frequency (TF/IDF) as these techniques could prove to be useful in the image classification realm as much as they are in the textual one.

Mix cluster with classification

6 References

- (see http://gendershades.org/) https://uk.mathworks.com/help/vision/examples/automatically-detect-and-recognize-text-in-natural-images.html - Models location: