Title of my document

Flavio R. de A. F. Mello

April 23, 2019

1 Domain and Task

2 Preprocessing

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".

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 224x224, 3 channel RGB image that was resized using bicubic interpolation.

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.

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.

For the improving the efficiency of label recognition, I set up a simple

./Images/labeling/samples.jpg

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)

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

Table 1: Group Image Labeling Through Clustering Progress

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.

./Images/labeling/cheatsheet.jpg

Figure 2: Labeling 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:

*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. Because the face extraction step will still select these faces when presented with a group image, and to avoid having these individuals being misclassified as one of the labeled ones, they were given arbitrary labels, which are then used to remove these entries when returning the P matrix in the RecognizeFace script. Figure **??** shows the individuals in question.

`./Images/labeling/unlabeled_inds.jpg`

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

*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.

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 3 different classification methods - Convolutional Neural

./Images/labeling/blurry.jpg

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

Networks (CNN), Support Vector Machines (SVM), and Random Forests (RF). The two 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 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.

3.6 Data Balance

3.7 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.

3.8 Training Process

3.9 Prediction Process

3.10 Initial Results

3.11 Tuning

3.12 Model Selection

4 Digit Recognition

4.1 Strategy

4.2 Results

5 Full Program

6 Conclusion

7 Further Work

- TF/IDF