Title of my document

Flavio R. de A. F. Mello

April 19, 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. Considering this study is for didactic purposes, it was decided to keep only every 3rd 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.

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 two substages: face detection and cropping.

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 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 to that they complement each other. 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). 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.

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 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 3.5 below.

3.1 HOG

Histogram of Oriented Gradients (HOG) 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 SURF

Based on scale-invariant feature transform (SIFT), the Speeded-Up Robust Features (SURF) algorithm is able to extract features in the form of image regions (i.e. blobs) that are invariant to scale. That is, a single feature represents the same visual blob at multiple different scales. This is done by repeatedly

- TF/IDF