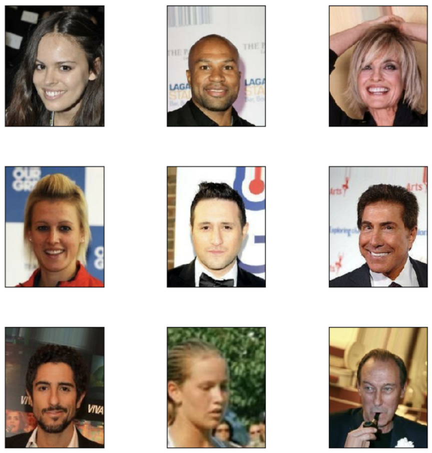
CelebA Attribute Prediction and Image Retrieval

This is the third assignment of a series of assignments as part of application for the JIO CV position. Here we will discuss facial attribute prediction on a CelebA dataset and how we can retrieve or cluster celeb images based on those predictions.

Introduction

Facial Attribute prediction is a Computer Vision (CV) task about deducing the set of attributes belonging to a face. Example attributes are the color of hair, hairstyle, age, gender, etc.

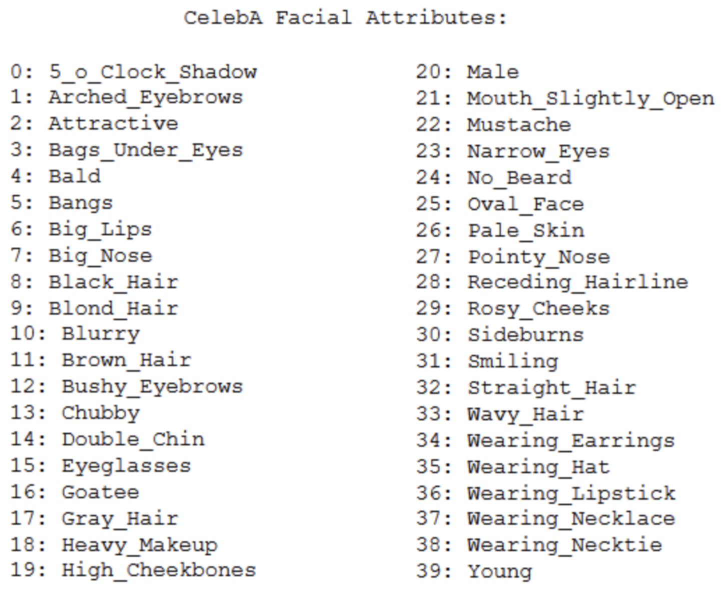


In general, facial attribute prediction is a challenging task: it involves face localization first, and then attribute prediction. Moreover, faces are inherently difficult to analyze due to their complex appearance. The appearance of a face can be altered, being even more complex, by face variations - *hairstyle, makeup, glasses, extreme lighting or extreme shadowing, emotions, pose* and so on.

Thus, it’s very important that a convolutional model is trained on difficult attributes of faces in order to generalize well to capture various attributes under any kind of conditions.

Data Exploration

CelebA dataset is a large-scale face attributes dataset with more than 200K celebrity images, covering a large amount of variations, each with 40 attribute annotations.



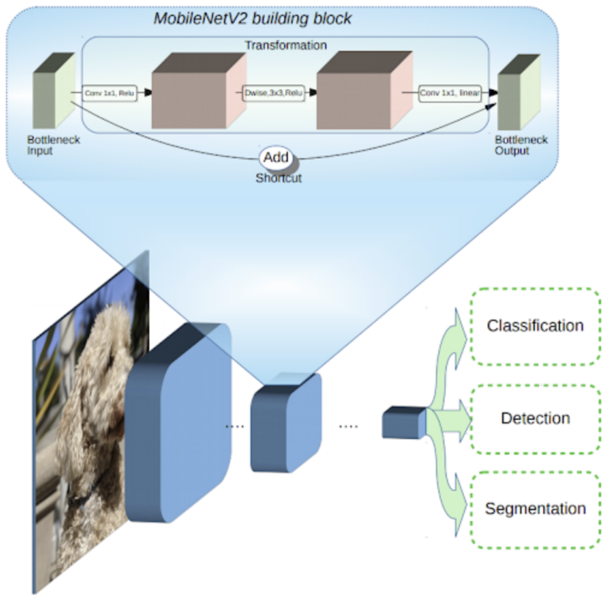
```
                 CelebA Facial Attributes:

0: 5_o_Clock_Shadow             20: Male
1: Arched_Eyebrows              21: Mouth_Slightly_Open
2: Attractive                   22: Mustache
3: Bags_Under_Eyes              23: Narrow_Eyes
4: Bald                         24: No_Beard
5: Bangs                        25: Oval_Face
6: Big_Lips                     26: Pale_Skin
7: Big_Nose                     27: Pointy_Nose
8: Black_Hair                   28: Receding_Hairline
9: Blond_Hair                   29: Rosy_Cheeks
10: Blurry                      30: Sideburns
11: Brown_Hair                  31: Smiling
12: Bushy_Eyebrows              32: Straight_Hair
13: Chubby                      33: Wavy_Hair
14: Double_Chin                 34: Wearing_Earrings
15: Eyeglasses                  35: Wearing_Hat
16: Goatee                      36: Wearing_Lipstick
17: Gray_Hair                   37: Wearing_Necklace
18: Heavy_Makeup                38: Wearing_Necktie
19: High_Cheekbones             39: Young
```

Each celeb image is associated with the above attribute, which can easily be converted to a vector of binary labels, where each attribute can be 0 or 1. By counting the times a 1 occurs for a certain attribute, it is possible to compute its absolute frequency (or even relative frequency).

Further we are also required to consider data imbalanced, which is quite evidence for celeb_a dataset, in which case the model can easily overfit the data points. All this problem can be mitigated by using a proper ***model, optimize and loss function***.

The Model

The model we are going to use is based on MobileNetV2 architecture, basically the same model but without the top classification layers (MobileNetV2 is built to output 1000 class probabilities).



First, the model takes one image (3 channels, 224×224 in size) at a time as input, and outputs a vector of probabilities of size n (40, one for each attribute). The i-th element of the vector is a real number between 0 (attribute i is not present) and 1 (attribute i is present).

Further, we had added following layers above MobileNetV2 model

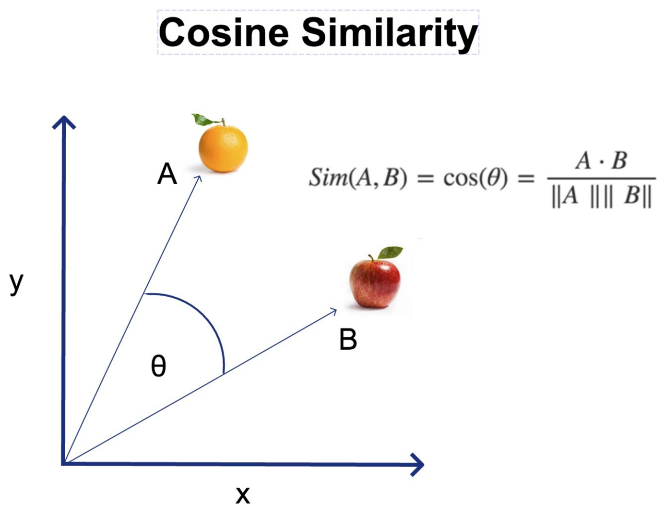
- `data_augmentation` layer : To argument input images to various random zoom, flip and shift inorder to increase the size of the training set.
- keras `preprocess_input` layer : To normalize the image between pixels -1 and 1 and reshape image from shape=(218, 178, 3) to shape=(224, 244, 3) as required by MobileNetV2.
- `GlobalAveragePooling2D`: To flatten the top layer to feed to FCL.
- `BatchNormalization and Dropout` : To reduce overfitting during training.

Optimizer, Loss Function and Metrics

For an optimizer, let's select the ***adadelta optimizer*** (adam optimizer is fine too) with default values: like adam, adadelta is an optimizer that requires little tuning because it is capable of automatically adapting the learning rate.

The **loss function**: responsible for monitoring the model's performance and guiding the optimization process. A good loss function for attribute prediction task, is a function able to discriminate the single elements of the attribute vector rather than considering the whole vector as a single object. The loss must understand that the vector [0, 1, 1, 0] is a lot different from this vector [1, 0, 0, 1].

Loss functions like Mean Absolute Error (MAE), Mean Square Error (MSE) or Root Mean Square Error (RMSE) are not capable of doing this kind of distinction: intuitively, they just count the 1s over a vector, and average on them, consequently losing information about position.



Instead, loss functions like ***binary cross-entropy and cosine similarity*** are the ideal choice.

Finally, we are going to use ***binary accuracy*** to measure performance of the model

Image Retrieval

After we extract (predicted) features from our query image, we calculate the distance between the query and all images. For doing this, we can use the ***Euclidean distance*** or l2 norm to measure it. If the number is getting smaller, the pair of images are similar to each other. The formula looks like this,

$$dist(q, img) = ||q - img||_2 = \sqrt{\sum_{i=1}^{n}(q_i - img_i)^2}$$

q (the query), img (all images), n (the number of feature vector elements), i (the position of the vector).

Next

Let's check out the code and visualize the result. I was able to achieve an accuracy of `0.5978` on my validation set, further hypertuning and epochs are required and also more free hardware resources 😞