Open Eyes Classifier

1 Problem Statements

Given the unlabeled dataset EyesDataset, one needs to train a classifier of open/closed eyes. The solution is implemented as a Python class named `OpenEyesClassifier` with `__init__(self)` and `predict(self, inplm)` methods. The former loads the model, and the latter outputs the probability-like score `is_open_score` from 0. to 1. (where 0. means eye is open and 1. – is closed) to the image file, set by relative or absolute path. It is suggested to assess the validation results with Equal Error Rate (EER) metric. EER must be no greater than 0.05. Note that test set is not available to examinee.

2 Labeling

There are several options of labeling data if bounded to only the dataset given.

1. manual labeling
2. clustering algorithm on the flattened data
3. extracting features with
   - an autoencoder (perceptual loss or just MSE)
   - or a network pretrained on a similar dataset,
   - or PCA/HOG/TSNE,

   then feature-based clustering

Let us briefly consider each of them. There are 4000 images in the dataset. If they were of high quality, it would be possible to label them in less than an hour. The point is that they are of small size (24x24); thus, in some cases, that makes it impossible even for a human to determine whether the eye is open or closed. Personal estimate of the share of such images in EyesDataset is around 20%. So, it is still possible to manually prepare the training data after a careful distillation if desired. In this work, I do not touch this matter, however, there are `rectify_labels` and its wrapper `mend_labels` in `src/data/utils.py` that may serve the goal if deployed in Jupyter Notebook.

As it has been mentioned, images are of small size, so one might expect decent results after direct application of `KMeans` on the flattened data. However, it is not the case here. Checking the labels with `mend_labels` confirms a more like random clustering. What we also can see from the score dynamics on the validation. Extracting features (with any approach stated above) should improve the situation, though it is not reflected in the experiments.

If we are allowed to use third-party datasets, an extra option emerges:

4. training a classifier on the data from almost identical distribution (MRL Eye Dataset), then labeling the target dataset with it

3 Classifier

It would be an overkill to use Inception-like architecture or pretrained ResNet-`XX` for EyesDataset. Considering a pretrained models, one should choose such a network that relies on a similar dataset. Either it is trained for feature extraction or refitted for classification on the target samples. With that said, fully convolutional network with batchnorms and ReLUs should be enough. The appropriate size for the receptive field is of a pupil's size or greater ($>$ 11x11). Image borders do not contain defining information; thence, padding may be omitted. One can think of adding skip connections: it will likely increase the expressiveness of the resulting model. I do not use them this time.

4 Experimental Setup

The plan is as follows. We train a simple classifier (`4*{Conv2d + BatchNorm2d + LeakyReLU} + Conv2d`) on MRL Eye Dataset, which I believe is the progenitor of EyesDataset (i.e., the latter is resulted from the former according to some sampling strategy). Then, we apply the model to form the labels of our target data. At this step, we need to eliminate low confidence samples from the dataset or/and apply label smoothing. Using undersampling method for MRL Eye Dataset and data augmentation techniques (like horizontal flip, brightness adjustment, small rotations $\sim 10°$, and maybe elastic distortion) for EyesDataset we could mix datasets in one. Additional fitting on the target samples can remove the covariance shift that could emerge during original sampling (EyesDataset creation).

5 Results

Tested clustering methods (feature extraction with PCA, HOG, and just flattened samples) have shown to be ineffective in labeling data. A significant percent of images are wrongly classified; therefore, training on such a dataset gets meaningless.

Training on a third-party dataset leads to more correct data markup. However, sometimes it becomes difficult to achieve the same validation score in the experiments with a fixed seed since random state changes with any code modification and also depends on the platform.