Gender classification by simple 7 layers CNN

Introduction

This work is a simple implementation of 7 layers CNN for gender binary classification. The project includes dataset and 4 python scripts, which could make the training, single image prediction, multi-image prediction with report of mAP, and create ground truth label file of format txt/npy. This CNN can get a good mAP within a quick training process.

Dataset preparation

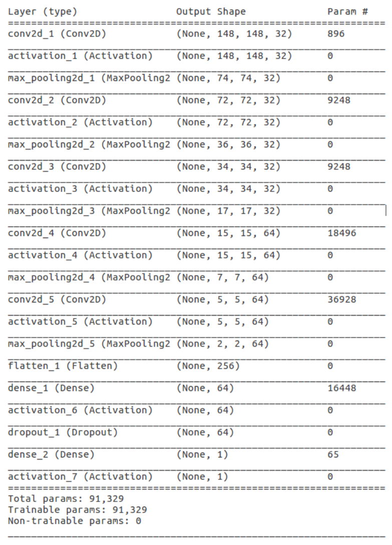
The original dataset is Large-scale CelebFaces Attributes (CelebA) Dataset, which is special for tasks of face identification. The part of images we will use is a extract of 575 photos. I divided this set into two, one part for training and another for validation and evaluation(mAP), to prevent form over-fitting and get an objective result.
In consideration of the size of dataset, some data augmentation tricks like shera mapping and zooming are tried.

Network structure and result

The object is to construct a simple and efficient CNN to make gender prediction. The both training and inference should be very quick as the requirement of advisor. To realize this task, the traditional CNN is a natural choice. At beginning, i designed a five layer CNN, with only 3 layers of 3x3 convolution to extract features and 2x2 pooling to change the dimension of feature maps. ReLU is used instead of sigmoid to avoid gradient vanishing. Fully connected layers is put at the end to make the binary prediction (or 2 classes classification). The mAP on validation set is about 92% by using this model.

To get a better result, I added 2 conditional layers basing on the former model. The new model(***Figure 1***) get a better result. The 2 additional layers don't make the training slower. On my computer (i7 6700hq, GTX960m, SSD), it takes about 2.2s to train an epoch of 500 images on mode GPU. It will not take more than 1 minute to train one epoch on CPU.

I record the loss and accuracy value (***Figure 2***) during the training to get the best trained weight.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 148, 148, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 72, 72, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 32)        9248
_________________________________________________________________
activation_3 (Activation)    (None, 34, 34, 32)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
activation_4 (Activation)    (None, 15, 15, 64)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 5, 5, 64)          36928
_________________________________________________________________
activation_5 (Activation)    (None, 5, 5, 64)          0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                16448
_________________________________________________________________
activation_6 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_7 (Activation)    (None, 1)                 0
=================================================================
Total params: 91,329
Trainable params: 91,329
Non-trainable params: 0
_________________________________________________________________
```

Figure 1

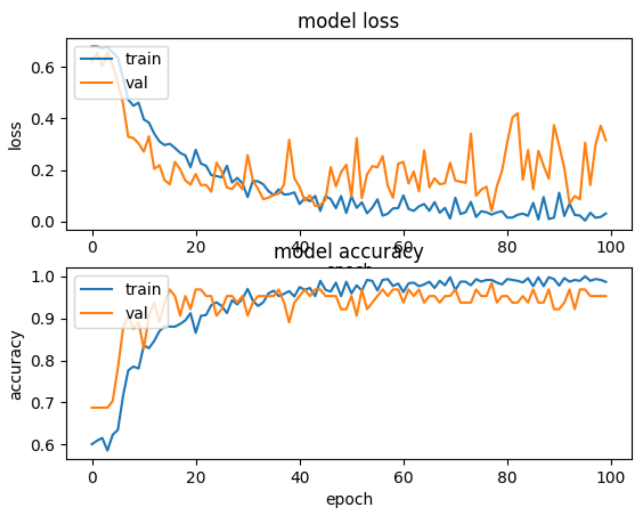
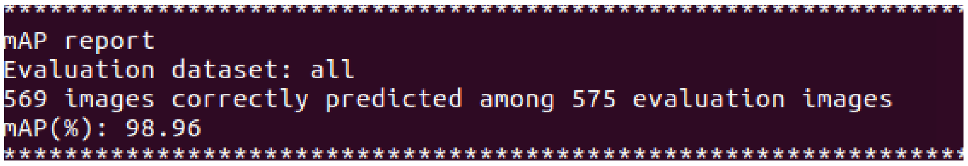


Figure 2

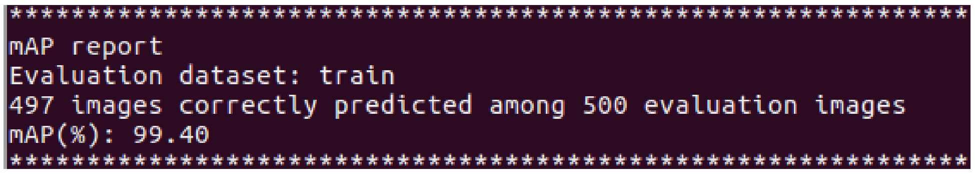
From the ***Figure 2,*** we can observe that overfitting begins at around epoch 60, where the loss of validation becomes difficult to converge. To get a better mAP on validation/evaluation dataset, we should take the weight file of epoch ±60.

I tested the weight of epoch 70 on whole dataset(***Figure 3***), training dataset(***Figure 4***) and validation dataset(***Figure 5***). It is clear that the model overfits on training set. With a more trained model, the mAP on training set can easily reach 100%, which makes no sense.



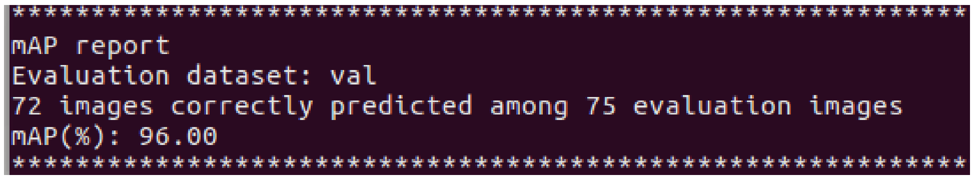
```
***************************************************************
mAP report
Evaluation dataset: all
569 images correctly predicted among 575 evaluation images
mAP(%): 98.96
***************************************************************
```

Figure 3



```
***************************************************************
mAP report
Evaluation dataset: train
497 images correctly predicted among 500 evaluation images
mAP(%): 99.40
***************************************************************
```

Figure 4



```
***************************************************************
mAP report
Evaluation dataset: val
72 images correctly predicted among 75 evaluation images
mAP(%): 96.00
***************************************************************
```

Figure 5

Residual nets are proved powerful for image classification. It overcomes the restriction of network depth. If I had more time, I would try to implement a simplified residual networks (layer already exists in Keras).

Manual for users

Requirement of libraries:
Keras (I tested with GPU support, CPU also should work without bug)
Thensorflow
Numpy
Python3 (Python2 not tested, possibly need minor modification. But we shouldn’t use python2 any more in 2018)

Following are optional:
opencv3
matplotlib

1. First of all, prepare your dataset as following structure: *root_project/all*, *root_project/train*, *root_project/val*. *all* means whole dataset, *train* means dataset for training, *val* means dataset for validation and evaluation. Then in each dataset create 2 sub folder '*man*' and '*woman*' and put the corresponding images in each folder.

2. (Optional) If you want, you can make a table of ground truth, by running

```
python3 dataset.py
```

You will find a txt and npy file in the folder 'all'. But it is not required for training or inference.

3. Traning. Run

```
python3 gender_train.py
```

You will get the saved model file (model.h5) and weight file (weightsxx.h5) at the project root folder.
In the code file **gender_train.py**, you can modify the epoch and batch size at **line 19**. Batch size =32 can get a good result. If have bug report of memory, please check this parameter.

4. Prediction on single image. Please run gender_single_predict -image_path -weight_path. For ex,

```
python3 gender_single_predict.py test1.jpg weights70.h5
```

5. Prediction on a group of images and calculate the mAP, run python3 gender_mAP.py -dataset_path -weight_path

```
python3 gender_mAP.py val weights70.h5
```