Training the face recognition model (s4_train_face_trainer)

Björn Bebensee (2019-21343)

December 3, 2020

0.1 Evaluate the trained network (Week 12) - Step 4

####Designed by Joon Son Chung, November 2020

This is based on https://github.com/joonson/face_trainer. You should read the code if you want to understand more about the training details.

In this step, we train the network on the generated dataset.

The baseline model, available from here, is trained on the VGGFace1 dataset using the softmax loss.

The training and validation sets should also be downloaded to the experiments folder `data_dir`.

`save_path` should be changed every time you run a new experiment.

The models take up a significant amount of disk space. Make sure that you have enough space on your Google Drive, and delete any unnecessary/ unsuccessful experiments.

[ ]:
```
from google.colab import drive
from zipfile import ZipFile
from tqdm import tqdm
import os, glob, sys, shutil, time
import torch
import torchvision.transforms as transforms
import torchvision.models as models
from PIL import Image
from datetime import datetime # used to timestamp the save_path to keep␣
 ↪checkpoints and logs

# mount Google Drive
drive.mount('/content/drive', force_remount=True)

# path of the data directory relative to the home folder of Google Drive
GDRIVE_HOME = '/content/drive/My Drive'
FOLDER      = 'University/2020 Fall/Machine Learning for Visual Understanding/
 ↪faces'

# specify paths
data_dir      = os.path.join(GDRIVE_HOME,FOLDER) ## path of the general␣
 ↪experiment
```

```
initial_model = os.path.join(GDRIVE_HOME,'University/2020 Fall/Machine Learning␣
 ↪for Visual Understanding/res18_vggface1_baseline.model') ## path to the␣
 ↪pre-trained model
train_zip     = os.path.join(data_dir,'train.zip') ## training data as zip
val_zip       = os.path.join(data_dir,'val.zip') ## validation data as zip
now = datetime.now().strftime("%Y%m%d_%H%M%S_") # get the current date and time␣
 ↪as e.g. 20201130_140220_
save_dir = now + 'experiment' # directory name for training logs and checkpoints
save_path     = os.path.join(data_dir, save_dir) ## training logs and trained␣
 ↪models will be saved here

# extract the cropped images
with ZipFile(train_zip, 'r') as zipObj:
  zipObj.extractall("/train_set")
with ZipFile(val_zip, 'r') as zipObj:
  zipObj.extractall("/val_set")
print('Zip extraction complete')
```

```
Mounted at /content/drive
Zip extraction complete
```

Make sure that the files have been extracted properly. Make sure that this is a reasonable number.

[ ]:
```
train_files = glob.glob('/train_set/*/*.jpg')
val_files   = glob.glob('/val_set/*/*.jpg')
print(len(train_files),'train set files and',len(val_files),'validation set␣
 ↪files found.')
```

```
58102 train set files and 2431 validation set files found.
```

First, clone the face recognition trainer from GitHub and add it to path.

[ ]:
```
!rm -rf face_trainer
!git clone https://github.com/joonson/face_trainer.git

sys.path.append('face_trainer')
```

```
Cloning into 'face_trainer'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 30 (delta 7), reused 25 (delta 5), pack-reused 0
Unpacking objects: 100% (30/30), done.
```

The training script. Please do not change, but try to read and understand.

[ ]:
```
import datetime
from utils import *
from EmbedNet import *
```

```
from DatasetLoader import get_data_loader
import torchvision.transforms as transforms

# ## ===== ===== ===== ===== ===== ===== ===== =====
# ## Trainer script
# ## ===== ===== ===== ===== ===== ===== ===== =====

def train_network(args):

    ## Make folders to save the results and models
    args.model_save_path     = args.save_path+"/model"
    args.result_save_path    = args.save_path+"/result"

    if not(os.path.exists(args.model_save_path)):
        os.makedirs(args.model_save_path)

    if not(os.path.exists(args.result_save_path)):
        os.makedirs(args.result_save_path)

    ## Load models
    s = EmbedNet(**vars(args)).cuda();

    ## Write args to scorefile
    scorefile = open(args.result_save_path+"/scores.txt", "a+");

    strtime = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    scorefile.write('%s\n%s\n'%(strtime,args))
    scorefile.flush()

    ## Input transformations for training
    train_transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Resize(256),
         transforms.RandomCrop([224,224]),
         transforms.RandomRotation(45), # added this transform
         transforms.ColorJitter(brightness=0.2, hue=0.2, saturation=0.2), #␣
↪added this transform
         transforms.RandomGrayscale(p=0.1), # added this transform
         transforms.RandomErasing(p=0.1, scale=(0.02, 0.1), ratio=(0.3, 3.3)), #␣
↪added this transform
         transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
↪225])])

    ## Input transformations for evaluation
    test_transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Resize(256),
```

```
        transforms.CenterCrop([224,224]),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
↪225])])

    ## Initialise trainer and data loader
    trainLoader = get_data_loader(transform=train_transform, **vars(args));
    trainer     = ModelTrainer(s, **vars(args))

    ## If initial model is specified, start from that model
    if(args.initial_model != ""):
        trainer.loadParameters(args.initial_model);
        print("Model %s loaded!"%args.initial_model);

    besteer   = 100
    bestmodel = ''

    ## Core training script
    for it in range(1,args.max_epoch+1):

        clr = [x['lr'] for x in trainer.__optimizer__.param_groups]

        print("Training epoch %d with LR %f "%(it,max(clr)));

        loss, traineer = trainer.train_network(trainLoader, verbose=True);

        if it % args.test_interval == 0:

            snapshot = args.model_save_path+"/model%09d.model"%it

            sc, lab = trainer.evaluateFromList(transform=test_transform,␣
↪**vars(args))
            result = tuneThresholdfromScore(sc, lab, [1, 0.1]);

            print("IT %d, VEER %2.4f"%(it, result[1]));
            scorefile.write("IT %d, VEER %2.4f\n"%(it, result[1]));

            trainer.saveParameters(snapshot);

            if result[1] < besteer:
                besteer   = result[1]
                bestmodel = snapshot

        print("TEER/TAcc %2.2f, TLOSS %f"%( traineer, loss));
        scorefile.write("IT %d, TEER/TAcc %2.2f, TLOSS %f\n"%(it, traineer,␣
↪loss));

        scorefile.flush()
```

```
    scorefile.close();

    print('Best validation EER: %2.4f, %s'%(besteer,bestmodel))
```

Specify the input arguments to the trainer below, and run to train the network. The validation losses will be printed below, and also saved to `save_path`.

See here for the meanings of each of these arguments, and see here for the list of available loss functions. If you use meta-learning loss functions, `nPerClass` must be 2 or more.

Note that the trainer includes a script to make sure that there are only `nPerClass` images per class per mini-batch. This helps with the meta-learning loss functions.

[ ]:
```
import easydict
args = easydict.EasyDict({ "batch_size": 256, # batch size
                           "trainfunc": "softmaxproto", # loss function
                           "lr": 1e-3, # learning rate
                           "lr_decay": 0.9, # how much to decrease the learning␣
 ↪rate, every 5 epochs
                           "weight_decay": 0, # regularization to reduce␣
 ↪overfitting (e.g. 1e-4 might be reasonable)
                           "margin": 0.2, # for AM-softmax and AAM-softmax
                           "scale": 30, # for AM-softmax and AAM-softmax
                           "nPerClass": 2, # support set + query set size for␣
 ↪meta-learning
                           "nClasses": 1007, #1010, # number of identities in the␣
 ↪training dataset
                           # Don't change below here!!
                           "save_path": save_path,
                           "max_img_per_cls": 500,
                           "nDataLoaderThread": 5,
                           "test_interval": 3,
                           "max_epoch": 15,
                           "optimizer": "adam",
                           "scheduler": "steplr",
                           "hard_prob": 0.5,
                           "hard_rank": 10,
                           "initial_model": initial_model,
                           "train_path": "/train_set",
                           "train_ext": "jpg",
                           "test_path": "/val_set",
                           "test_list": "/val_set/test_list.csv",
                           "model": "ResNet18",
                           "nOut": 512,
                           "mixedprec": False})

train_network(args)
```

```
Initialised Softmax Loss
Initialised AngleProto
Initialised SoftmaxPrototypical Loss
58102 files from 1007 classes found.
Initialised Adam optimizer
Initialised step LR scheduler
__L__.fc.weight is not in the model.
__L__.fc.bias is not in the model.
Model /content/drive/My Drive/University/2020 Fall/Machine Learning for Visual
Understanding/res18_vggface1_baseline.model loaded!
Training epoch 1 with LR 0.001000
Processing (24320) Loss 8.816987 TEER/TAcc 20.600% - 166.03 Hz
TEER/TAcc 20.60, TLOSS 8.816987
Training epoch 2 with LR 0.001000
Processing (24320) Loss 5.123374 TEER/TAcc 61.589% - 166.11 Hz
TEER/TAcc 61.59, TLOSS 5.123374
Training epoch 3 with LR 0.001000
Processing (24320) Loss 4.172480 TEER/TAcc 73.487% - 166.07 Hz
Reading 2400 of 2431: 70.43 Hz, embedding size 512
Computing 46500 of 46589: 17344.58 Hz
IT 3, VEER 10.8638
TEER/TAcc 73.49, TLOSS 4.172480
Training epoch 4 with LR 0.000900
Processing (24320) Loss 3.712751 TEER/TAcc 78.727% - 147.75 Hz
TEER/TAcc 78.73, TLOSS 3.712751
Training epoch 5 with LR 0.000900
Processing (24320) Loss 3.445094 TEER/TAcc 81.856% - 147.83 Hz
TEER/TAcc 81.86, TLOSS 3.445094
Training epoch 6 with LR 0.000900
Processing (24320) Loss 3.243493 TEER/TAcc 83.657% - 143.87 Hz
Reading 2400 of 2431: 70.04 Hz, embedding size 512
Computing 46500 of 46589: 14052.55 Hz
IT 6, VEER 10.2832
TEER/TAcc 83.66, TLOSS 3.243493
Training epoch 7 with LR 0.000810
Processing (24320) Loss 3.067869 TEER/TAcc 85.504% - 146.53 Hz
TEER/TAcc 85.50, TLOSS 3.067869
Training epoch 8 with LR 0.000810
Processing (24320) Loss 2.933756 TEER/TAcc 86.766% - 145.73 Hz
TEER/TAcc 86.77, TLOSS 2.933756
Training epoch 9 with LR 0.000810
Processing (24320) Loss 2.808039 TEER/TAcc 87.730% - 146.42 Hz
Reading 2400 of 2431: 69.81 Hz, embedding size 512
Computing 46500 of 46589: 13626.97 Hz
IT 9, VEER 9.8344
TEER/TAcc 87.73, TLOSS 2.808039
Training epoch 10 with LR 0.000729
Processing (24320) Loss 2.697170 TEER/TAcc 88.984% - 146.07 Hz
```

```
TEER/TAcc 88.98, TLOSS 2.697170
Training epoch 11 with LR 0.000729
Processing (24320) Loss 2.592930 TEER/TAcc 89.745% - 146.69 Hz
TEER/TAcc 89.75, TLOSS 2.592930
Training epoch 12 with LR 0.000729
Processing (24320) Loss 2.549067 TEER/TAcc 90.171% - 145.17 Hz
Reading 2400 of 2431: 71.32 Hz, embedding size 512
Computing 46500 of 46589: 14418.99 Hz
IT 12, VEER 9.6728
TEER/TAcc 90.17, TLOSS 2.549067
Training epoch 13 with LR 0.000656
Processing (21760) Loss 2.454362 TEER/TAcc 91.045% - 124.69 Hz
```

First, evaluate the baseline model yourself. (The baseline model has been trained with this trainer using the softmax loss on the images of the VGGFace dataset cropped to the same specification as our dataset.) Then, try various experiments and record the results. You can try as many or few experiments as you like.

Optional: try to see if you can get better performance by altering the data augmentation. In the current structure this will need to be changed inside `train_network` function.

The parameters in the table denote **changes** from the default parameters that were given with the notebook.

#	Params	Val EER	Notes
0	baseline trained on vggface	22.35	
1	softmax / batch_size=256	11.96	
2	softmax / batch_size=256 / transforms: rotate(45), jitter(0.1), greyscale(0.1), erase(p=0.5,[0.01,0.1])	11.60	
3	softmax / batch_size=200	11.75	
4	softmax / batch_size=100	11.83	
5	softmax / batch_size=100 / weight_decay 1e-4	13.25	Surprisingly performing worse
6	softmax / batch_size=100 / weight_decay=1e-4 / lr=0.0001	12.15	Underfits due to low lr
7	aamsoftmax / batch_size=256 / s=30 m=0.1	10.93	
7	aamsoftmax / batch_size=256 / s=30 m=0.2	11.08	
8	aamsoftmax / batch_size=256 / s=30 m=0.3 / weight_decay=5e-5	10.92	
9	amsoftmax / batch_size=256 / s=30 m=0.1	11.63	
10	angleproto / batch_size=128 / nPerClass=2	10.09	
11	angleproto / batch_size=64 / nPerClass=4	10.53	
12	softmaxproto / batch_size=256 / nPerClass=2	9.82	
13	softmaxproto / batch_size=256 / nPerClass=2 / lr_decay=0.5	9.81	
14	softmaxproto / batch_size=256 / nPerClass=2 / lr_decay=0.5 / weight_decay=5e-4	10.58	

#	Params	Val EER	Notes
15	softmaxproto / batch_size=256 / nPerClass=2 / lr_decay=0.5 / transforms: rotate(45), jitter(0.2), greyscale(0.1), erase(p=0.2,[0.05,0.15])	9.78	Model underfits, try with higher initial lr or lower decay, possibly less erasing (too difficult)
16	softmaxproto / batch_size=256 / nPerClass=2 / lr_decay=0.9 / transforms: rotate(45), jitter(0.2), greyscale(0.1), erase(p=0.1,[0.02,0.1])	9.67	Still underfits, could not finish training in time (score as of epoch 12)
17	ge2e / batch_size=256 / nPerClass=2	10.31	
18	triplet / batch_size=256 / nPerClass=2	10.45	

Choose the best model based on validation performance, and test on `test.zip` and `test_foreign.zip` using Step 5 of the code. Write the results below.

Evaluation on the test set:

```
Baseline model:
test: EER 20.9102
test_foreign: EER 16.4218

Model with best validation performance (#16):
test: EER 8.1117
test_foreign: EER 6.8275
```

If you have any other thoughts or feedback, please write here.

Although test_foreign is not perfectly "in-distribution", it seems finetuning on our Korean faces dataset has not only narrowed the gap between the test sets but significantly improved performance on test_foreign as well. Clearly the initial baseline model's architecture is capable of performing much better on Asian faces indicating that it had been trained on a very biased dataset which includes only very few Asian faces.
Unfortunately I think the time was somewhat constrained so I did not manage to test as extensively with GE2E and Triplet losses as they were somewhat lower on my priority list after seeing results in the AM-Softmax, AAM-Softmax and Angular Prototypical Loss papers. It was not al-

ways possible to run multiple experiments at once due to memory constraints (I think Colab caps this across notebooks?).
I think it could have been interesting to test on the original test set of faces as well (that the baseline was trained on) just to see how much the model is actually forgetting about faces not in the distribution during finetuning.