TASK 1: Improve The Performance On AFLW2000 Dataset.

Overview

Methodology

In order to improve the test score on AFLW2000 dataset, there are several possible approaches. Fundamentally changing the network structure could be the most effective way, but it is also quite involved. Alternatively, optimizing the current model and training strategies is more straight forward. Therefore, I propose the following methods to improve the performance.

Strategies

Training Strategies

Personally, I think the official implementation of HopeNet can not really be regarded as a good one. A few important components of conventional deep learning models, such as valiation and overfitting prevention, are missing in it. Thus, I suggest that the two parts below can be added into the training process,

- Training / valiation split.
- Learning rate decay.

Training / valiation split is one of the most widely used strategy to prevent ML models from overfitting. ML methods with massive trainable weights are likely to be overfitted on the training set, which means it will only perform well on the training set, but generalize poorly on the testing set. In order to prevent the model from overfitting, one could split the dataset into training set and validation set. The later dataset is used for monitoring the performance of the model. When the valiation loss starts increasing while the training loss keeps decreasing, the model is likely to be overfitted, and the training needs to be stopped.

Learning rate decay is one of the most common practice to training ResNet, which is the backbone of HopeNet. Learning rate is exponentially decayed when the training loss reaches a plateau. An example of its effect can be found in Figure 1 in Kaiming He's ResNet paper.

Network Architecture

Although it is not easy to come up with some brand-new ideas regarding network architecture, there are still some simple-yet-effective methods that we could adopt in this model. One common practice is to add

attention module in the CNN, and there are couples of such approaches available, such as channel-wise attention and spatial attention. Here, I adopt the lately introduced attention module, CBAM. In the paper, the author reported a clear increase in performance after adding the CBAM module into ResNet-50. Thus, I also expect that CBAM can help with improving the performance of the ResNet-50 used in this head-pose estimation task.

Implementation

Coding Convention

I found the official implementation of HopeNet didn't following any coding convention, and most scripts in it can't pass the flake8 check. Thus, I re-rewrote some scripts, following the google coding style.

Train / Validation Split

In my implementation, the 300W_LP dataset is randomly split into two sets by a train-to-valiation ratio of 9:1.

Learning Rate Decay.

The initial learning rate is set to 1e-5, and the model will be trained for 25 epochs in total, as suggested in the paper. In my implementation, after the 8th epoch, the learning rate will be set to 1e-6. And after the 18th epoch, it will be set to 1e-7.

CBAM

Implementation of CBAM in this repo is modified and reused in my code.

Filename List

As the author didn't provide the script to produce filename_list.txt, I wrote it by myself.

How To Use

Download dataset.

AFLW2000 and 300W_LP can be downloaded here.

```
mkdir data
cd data
unzip AFLW2000-3D.zip
unzip 300W-LP.zip
```

Dependencies

python == 3.5 numpy == 1.16.1 opencv-python == 4.0.0.21 torch = 1.0.1 torchvision == 0.4.0+cu92 pandas == 0.24.1 Pillow == 5.4.1 scipy == 1.2.0

NOTE Since I only have a GPU with cuda-9.0, the pytorch version 1.0.1 is specially compiled for this cuda version. I guess the code is also compatible with other cuda & pytorch versions, but I didn't test it.

Preprocess

To produce the filename_list.txt, run the command below,

```
python filename_generator.py --dataset DATASET --dataset_path PATH
```

DATASET ∈ {AFLW2000, 300W-LP}. PATH ∈ {./data/AFLW2000, ./data/300W-LP}

a text file named 'filename_list.txt' will be stored in corresponding dataset.

NOTE 31 images will be eliminated in AFLW2000 dataset, as suggested in the paper.

Download Pre-trained Model.

Pre-trained model of my approach can be downloaded via google drive link.

```
unzip checkpoints.zip
```

Train

```
bash train.sh
```

NOTE please modify the hype-parameters in train.sh accordingly before starting training.

Test

```
bash test.sh
```

NOTE please modify the hype-parameters in test.sh accordingly before starting testing.

Reproduce My Results

```
python test_attnet.py \
--gpu 0 \
--dataset AFLW2000 \
--batch_size 32 \
--data_dir ./data/AFLW2000/ \
--filename_list ./data/AFLW2000/filename_list.txt \
--snapshot ./checkpoints/resnet.pkl
```

NOTE checkpoint.pkl is the pre-trained checkpoint downloaded via the google drive link mentioned above.

Results & Comparison

The proposed model is trained for 25 epochs with initial learning rate 0.00001, batch size 32. **NOTE** that the scores of the original HopeNet are directly from the paper. According to the response of the author on github, he used batch size 128 to training the HopeNet. However, since I only have a GPU with 8GB RAM, training with batch size 128 will invoke 'out of memory' error. Thus, I only trained my method with a batch size of 32. I believe that performance of my method can be further boosted with a larger batch size.

Model	Yaw	Pitch	Roll	MAE
Mine ($\alpha = 2$)	**5.275**	**6.138**	**4.852**	**5.422**
HopeNet ($\alpha = 2$)	6.470	6.559	5.436	6.155

As shown in the table above, my method outperforms the original HopeNet with a great margin in every aspect.

TASK 2: Simplify The Backbone

Overview

There are couples of solutions available out there about the simplification of CNN architecture. Several candicates are:

- MobileNet https://arxiv.org/abs/1704.04861
- ShuffleNet https://arxiv.org/abs/1707.01083
- CondenseNet https://arxiv.org/abs/1711.09224
- EfficientNet https://arxiv.org/abs/1905.11946
- ...

All of them are relatively lighter than ResNet and VGG-Net, with negligible loss of performance.

Here, I conducted experiment on ShuffleNet V2.

Comparison Between ResNet-50 and ShuffleNet V2

Model	#Params	Checkpoint Size
ResNet-50	26.5M	112MB
ShuffleNet V2	2.5M	11MB

Results

Backbone	Yaw	Pitch	Roll	MAE
ShuffleNetv2 ($\alpha = 2$)	9.004	7.527	6.967	7.562
ResNet-50 ($\alpha = 2$)	**5.275**	**6.138**	**4.852**	**5.422**

Download Pre-trained Model.

NOTE This step can be ignored if you have already downloaded checkpoints in the first task, as two checkpoints are downloaded together.

If not, go ahead.

Pre-trained model of my approach can be downloaded via google drive link.

```
unzip checkpoints.zip
```

Reproduce My Results

```
python test_shufflenet.py \
--gpu 0 \
--dataset AFLW2000 \
--batch_size 32 \
--data_dir ./data/AFLW2000/ \
--filename_list ./data/AFLW2000/filename_list.txt \
--snapshot ./checkpoints/shufflenet.pkl
```