AI6126: Advanced Computer Vision

Li Xiaochen (G2102142D)

Li0029en@e.ntu.edu.sg

Project 1: CelebAMask Face Parsing

Background

The objective of this challenge is to design and train a face parsing network which assigns pixel-wise labels for each semantic components such as eyes, nose, mouth and so on. The dataset used is from the mini-dataset from CeleAMask-HQ Dataset[1], which contains 5000 training and 1000 validation pairs of images, and both images and annotations have a 512x512 resolution.

The performance of this face parsing network will be evaluated based on the mIoU between the predicted masks and the ground truth of the test set which contains 1000 face images.

Model

For the model of this face parsing network, we used RestNet50_v1c as the pretrained backbone. The ImageNet pretrained model was loaded at the beginning. The depth of backbone is 50 and number of stages is 4. The style of backbone is 'pytorch' which means that stride 2 layers are in 3x3 conv. The type of decode head is 'DepthwiseSeparableASPPHead' and the type of auxiliary head is 'FCNHead'. The type of loss function is Cross Entropy Loss. And we use SGD as the optimizer in the config. The batch size is set to 4 and the max iteration is 30000.

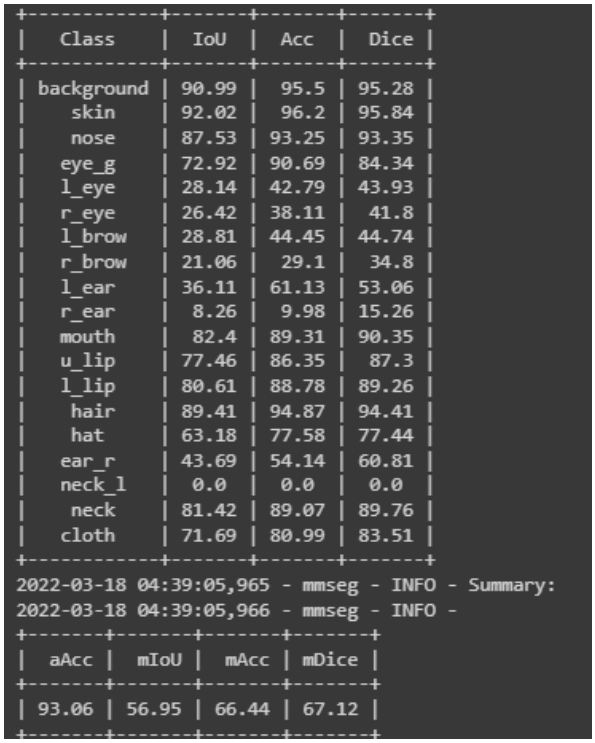
The Loss Function

For this face parsing network, we use Cross Entropy Loss as the loss function. The formular for Cross Entropy Loss is given by:

$$loss = -\sum_{k}^{K} y^{(k)} log\hat{y}^{(k)}$$

Cross Entropy Loss will calculate the average difference between the actual and predicted probability distributions for all classes in the problem. Here $y^{(k)}$ is 0 or 1, indicating whether class label k is the correct classification.

During the first run we can notice that, after 20k iterations, some of the class show very low IoU results, especially the neck_l (necklace) class (Figure 1).



Class	IoU	Acc	Dice
background	90.99	95.5	95.28
skin	92.02	96.2	95.84
nose	87.53	93.25	93.35
eye_g	72.92	90.69	84.34
l_eye	28.14	42.79	43.93
r_eye	26.42	38.11	41.8
l_brow	28.81	44.45	44.74
r_brow	21.06	29.1	34.8
l_ear	36.11	61.13	53.06
r_ear	8.26	9.98	15.26
mouth	82.4	89.31	90.35
u_lip	77.46	86.35	87.3
l_lip	80.61	88.78	89.26
hair	89.41	94.87	94.41
hat	63.18	77.58	77.44
ear_r	43.69	54.14	60.81
neck_l	0.0	0.0	0.0
neck	81.42	89.07	89.76
cloth	71.69	80.99	83.51

2022-03-18 04:39:05,965 - mmseg - INFO - Summary:
2022-03-18 04:39:05,966 - mmseg - INFO -

aAcc	mIoU	mAcc	mDice
93.06	56.95	66.44	67.12

Figure 1. Summary for the 1st run on val set

In order to solve this issue, two optimization methods were tried in the training model. The first one is Class Balanced Loss, for dataset that is not balanced in classes distribution, we may change the loss weight of each class, the class_weight will be passed into CrossEntropyLoss as weight argument. The example class_weight setup for this face parsing network is shown as below (Figure 2):

```
loss_decode=dict(
    type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0,
    class_weight = [0.8, 0.8, 0.8, 1.14, 1, 1, 1.14, 1.14, 1.14, 1.14,
                    1, 1, 1, 0.8, 1.2, 1.25, 1.5, 1, 1.15])),
```

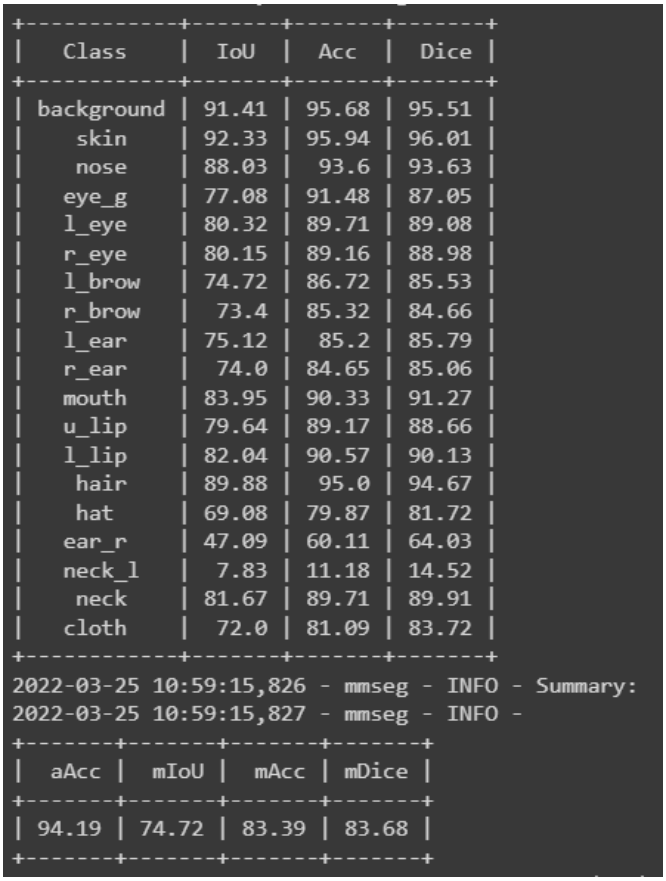
Figure 2. class_weight setup for CrossEntropyLoss

The second method is Online Hard Example Mining (OHEM) (Figure 3), in this way, only pixels with confidence score under 0.7 are used to train. And We keep at least 100000pixels during training.

```
sampler=dict(type='OHEMPixelSampler', thresh=0.7, min_kept=100000),
```

Figure 3. Online Hard Example Mining

After implemented both methods above, the final IoU of each class become (figure 4), the IoU of neck_l has been improved from 0 to 7.83:



Class	IoU	Acc	Dice
background	91.41	95.68	95.51
skin	92.33	95.94	96.01
nose	88.03	93.6	93.63
eye_g	77.08	91.48	87.05
l_eye	80.32	89.71	89.08
r_eye	80.15	89.16	88.98
l_brow	74.72	86.72	85.53
r_brow	73.4	85.32	84.66
l_ear	75.12	85.2	85.79
r_ear	74.0	84.65	85.06
mouth	83.95	90.33	91.27
u_lip	79.64	89.17	88.66
l_lip	82.04	90.57	90.13
hair	89.88	95.0	94.67
hat	69.08	79.87	81.72
ear_r	47.09	60.11	64.03
neck_l	7.83	11.18	14.52
neck	81.67	89.71	89.91
cloth	72.0	81.09	83.72

2022-03-25 10:59:15,826 - mmseg - INFO - Summary:
2022-03-25 10:59:15,827 - mmseg - INFO -

aAcc	mIoU	mAcc	mDice
94.19	74.72	83.39	83.68

Figure 4. Summary for the best result on val set

Optimizer and Scheduler

SGD optimizer with momentum is used for this experiment. The learning rate is 0.01, momentum is 0.9 and the weight decay is set to 1e-4.

In order to make the model converge in a faster way, the CyclicLrUpdater and Cyclic MomentumUpdater are used to modify model's momentum according to learning rage. The code is shown as below:

```
lr_config = dict(policy='cyclic', target_ratio=(10, 1e-4), cyclic_times=1, step_ratio_up=0.4)
momentum_config = dict(policy='cyclic', target_ratio=(0.85 / 0.95, 1), cyclic_times=1, step_ratio_up=0.4)
```

Figure 5. momentum schedule

Training Curves

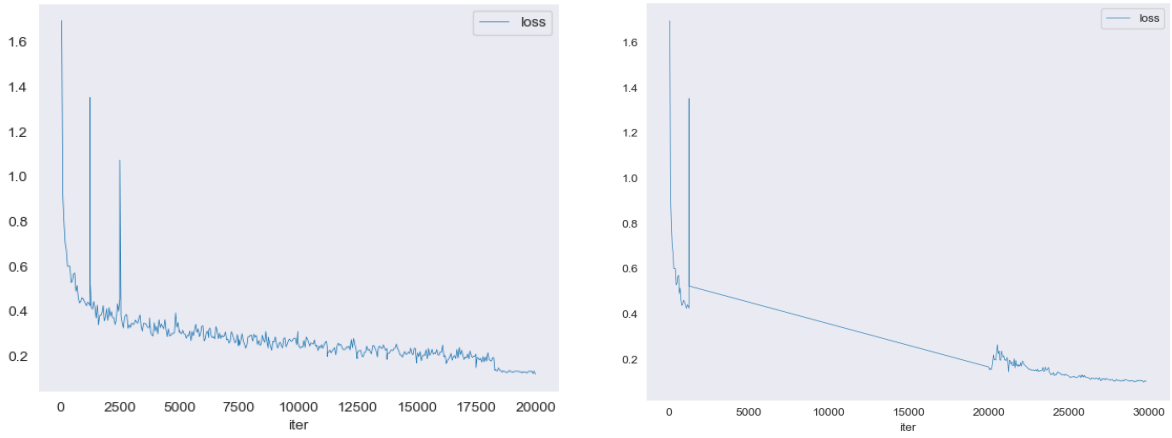


Figure 6. loss curve for val set (left: 20000 iters; right: 30000 iters resume from 20000)

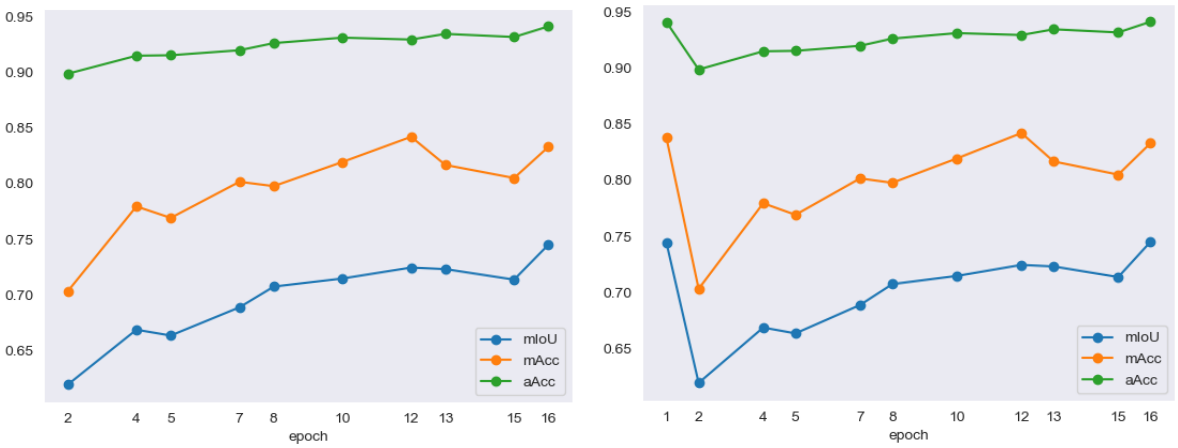


Figure 7. mIoU, mAcc, aAcc curve for val set

(Left: 20000 iters; right: 30000 iters resume from 20000)

Figure 6 and 7 shows the training curve of loss, mIoU, mAcc and aAcc on the validation set. The loss was reduced from 1.969 to 0.1025, and the mIoU was increased from 57.18 to 74.72 after 20000 iterations.

Number of Parameters

The number of parameters of this model is 43589878.

```
parameters = sum(p.numel() for p in model.parameters() if p.requires_grad)
print(parameters)
```

```
43589878
```

Training machine

For this project, both training and testing ran on Google Colab. The information of the training machine is shown as below:

```
# Check nvcc version
!nvcc -V
# Check GCC version
!gcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:09:46_PDT_2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.TC455_06.29190527_0
gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

```
!nvidia-smi
```

```
Fri Mar 25 12:09:30 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   59C    P0    29W /  70W |   9288MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

Test Result

The best mIoU on the 1000 test image is 74.25

74.2503863898	test_result-20220325T110734Z-001-v6-3w.zip	03/25/2022 11:09:59	164926	Finished	✔	✚

References

[1] MMSegmentation Tutorial
https://mmsegmentation.readthedocs.io/en/latest/tutorials/config.html