CE7454 Project 1 Report
CelebAMask Face Parsing

Teerameth Rassameecharoenchai
Nanyang Technological University
50 Nanyang Ave, Singapore 639798
C240196@e.ntu.edu.sg

Abstract

This project addresses the challenge of face parsing using the CelebAFaceMask Mini-Dataset, a subset of the CelebAMask-HQ Dataset containing 5000 training pair of high-resolution images and segmentation masks. The task involves developing a model with fewer than 2,000,000 trainable parameters to predict segmentation masks for 19 different facial classes. The dataset presents significant challenges due to its highly skewed class distribution, with the top 3 classes accounting for 86% of the data. To tackle this problem, I propose a model architecture based on MobileNetV3-small as the backbone, along with depth-wise separable convolutions, atrous convolutions, and atrous spatial pyramid pooling (ASPP) to capture multi-scale context while maintaining a low parameter count. The model is optimized using the Lovász-Softmax loss function to directly optimize the Intersection over Union (IoU) metric and handle class imbalance. I also employ various regularization techniques, including data augmentation and dropout, to prevent overfitting. Hyper-parameter tuning is performed using Bayesian optimization to maximize the mean IoU (mIoU) on the validation dataset.

Class	Count	Percentage
0	449,935,823	28.79%
1	398,822,925	25.52%
2	32,356,677	2.07%
3	4,068,509	0.26%
4	3,507,146	0.22%
5	3,502,365	0.22%
6	6,645,111	0.43%
7	6,494,166	0.42%
8	7,316,313	0.47%
9	6,005,039	0.38%
10	4,687,025	0.30%
11	6,505,249	0.42%
12	10,672,967	0.68%
13	494,500,447	31.64%
14	15,665,394	1.00%
15	3,730,286	0.24%
16	227,954	0.01%
17	64,725,027	4.14%
18	53,495,577	3.42%
Total	1,562,364,000	100.00%

Table 1. Distribution of Labels

1. Introduction

1.1. CelebAFaceMask Mini-Dataset

A mini-dataset for this project came from CelebAMask-HQ Dataset [4] containing 5000 training and 1000 validation pairs of images and segmentation masks, where both images and annotations have resolution of 512x512. The annotation contain 19 difference classes including all facial components and accessories. Table 1 show annotations count and distribution for each classes from both training and validation datasets.

This distribution shows highly skewed distribution with top 3 classes (13, 0, 1) account for 86% of data. The class 16 is rarest such that ratio of most common to rarest class is 3164:1. This distribution presents significant challenges for getting high mIoU and need some strategies to address the class imbalance.

1.2. Goals and Constraints

This project require model to contain fewer than 2,000,000 trainable parameters. The model also required to be trained from scratch without external dataset and pretrained weight. The final score for ranking come from mIoU between the predicted masks and the ground truth of the test set.

2. Model

Since the images size and segmentation mask are relatively large compared to model's size limitation. A model us-

ing depth-wise separable convolution was selected for the backbone network due to it's low parameter count while still maintaining good performance. To predict a segmentation mask this large, I decide to use atrous convolution and atrous spatial pyramid pooling for capturing multi-scale context to improve localization performance which is a downside of the max-pooling and down-sampling in typical DCNNs.

2.1. Backbone

The small variant of MobileNetV3 [3] was selected to use as a backbone template in this project. The original MobileNet-v3-Small has 12 blocks (excluding classifier and polling layers) and 2,542,856 trainable parameters which slightly greater than the constrain. Therefore, I pick only first (12 - *backbone_removed_layers*) layers to use as a backbone network to reduce the parameter count. While *backbone_removed_layers* is number of removed layered which is configurable during hyper-parameter tuning. The model API from PyTorch library provide configuration parameter *width_mult* which is a multiplier that affects the number of channels of the model. This parameter was used to adjust overall model size during hyper-parameters tuning to maximize the model size to get as close but still lower than 2,000,000 using binary search. During each iteration, the binary search will build the model using provided architecture related hyper-parameters (e.g. *base_atrous_rate*, *backbone_removed_layers*) then propose a candidate *width_mult* parameter until it considered optimized. If given hyper-parameters can't be used to build a model that met the requirements (e.g. over parameterized) it will specify validation loss as infinity during hyper-parameter tuning. This result in scalable backbone that can adjust itself to fill up number of parameters when combined with other parts of the model which may have various size from difference hyper-parameters.

2.2. Atrous Convolution

Typical DNNs use encoder which typically repleted max-pooling and stride layers to extract feature from the image but result in smaller feature map after each operation. To due with relatively large segmentation mask prediction (same size as input), I decided to replace it with implement atrous convolution and atrous spatial pyramid pooling (ASPP) [2]. Atrous (Dilated) convolution is a modified version of standard convolution that introduces spaces between the kernel elements. (**r - 1**) zero(s) will be inserted between each successive values inside a kernel along each spatial dimension, where **r** is *atrous_rate*. For instance, with a dilation rate of 2, a 3x3 kernel would effectively cover a 5x5 area of the input, with only 9 elements being used for computation (the rest are skipped). This allows the network to capture a larger receptive field without increasing the number of parameters or the amount of computation.

2.3. Atrous Spatial Pyramid Pooling (ASPP)

ASPP is a module designed to capture multi-scale information in semantic segmentation tasks. It applies multiple parallel atrous convolutions with different atrous rates to the input feature map before concatenation of all these features together. Then applies a final 1x1 convolution to reduce the channel dimension. The configurable ASPP module was implemented as a class called **ConfigurableASPP()** inside *model.py* consist of 5 main components.

1. *conv1*: 1x1 convolution branch
2. *conv2*: 3x3 depth-wise separable convolution branch
3. *atrous_branches*: Multiple atrous convolution branches with different dilation rate for captures context at different scales. This branch also use depth-wise separable convolutions for efficiency.
4. *global_avg_pool*: Global average pooling branch for captures global context information
5. *output_conv*: Final 1x1 convolution to combines and processes the outputs from all branches

The *forward* method feed input tensor thought *conv1* and *conv2*. Then applies each atrous convolution branch and global average pooling in parallel . Finally, apply the final output convolution to combines and processes the outputs from all branches.

This module also accept 4 configurations for hyper-parameter tuning

- *in_channels*: Number of input channels. Automatically calculated after achieve optimized value of *backbone_width_multiplier* during binary search.
- *out_channels*: Number of output channels. Specified by *aspp_output_channels* parameter during hyper-parameters tuning. (selected from choices of [128, 256, 512])
- *base_rate*: Base number for atrous rate. Specified by *base_atrous_rate* parameter during hyper-parameters tuning. (uniformly sampled within range of [2, 8])
- *atrous_depth*: Number of multiplier used for extending atrous rate. Specified by *artrous_depth* parameter during hyper-parameters tuning. (uniformly sampled within range of [2, 6])

For example, with *base_rate* = 4 and *atrous_depth* = 3 it will give *atrous rates* of [4*1, 4*2, 4*3] = [4, 8, 12]. This will create 3 branches of atrous convolution with 3 difference dilation rates.

2.4. Putting Together

A class **MobileNetV3ASPP()** from *model.py* presenting the whole model used in this project. This class combine

1. A portion of MobileNetV3-small architecture as backbone
2. A ConfigurableASPP module

3. A final classification layer for producing the final class predictions for each pixel

The *forward* method then feed input tensor thought these three modules before up-samples the output to match the input resolution.

I also put a function called *_init_weights()* to initializes the network's weights and biases based on layer's type for prevent vanishing/exploding gradients.

1. For convolutional layers (*nn.Conv2d*):
   - Weights: Kaiming normal initialization (suited for layers with ReLU activations)
   - Biases: Set to zero
2. For batch normalization layers (*nn.BatchNorm2d*):
   - Weights: Set to one
   - Biases: Set to zero

3. Optimization

3.1. Loss Function

Since classes in dataset of this project is heavily imbalanced, using categorical cross entropy (CCE) is not a good idea for maximizing mIoU of the prediction. CCE tends to favor the majority class, which can lead to poor IOU on minority classes. I decide to use *Lovász-Softmax loss* [1] which designed to directly optimize the Intersection over Union (IoU) metric. Lovász-Softmax loss was reported to better handles imbalanced classes in segmentation tasks and provides smoother gradients compared to direct IoU optimization. I imported this calculation method directly from author's repository as *lovasz_losses.py*. The Lovász-Softmax loss can be obtained by calling *lovasz_losses.lovasz_softmax()* method in the same way as other PyTorch's built-in criterion.

3.2. Learning Rate Scheduler

The training use Adam optimizer using fixed batch size of 8 with configurable and choices of difference *scheduler* for hyper-parameter tuning. Scheduler choices:

1. *CosineAnnealingLR*: adjusts the learning rate according to a cosine function, need to specify total *epochs* to correctly scale the learning rate from base learning rate *base_lr*.
2. *ReduceLROnPlateau*: reduces the learning rate when a specified metric (Lovász-Softmax loss) has stopped improving.
3. *LinearLR*: decreases the learning rate linearly from the base learning rate *base_lr* across specified total *epochs*.

The training loop will be executed for *epochs* times which is a hyper-parameter controlling number of maximum epoch during training.

3.3. Early Stopping

The validation dataset was used at the end of each epoch to track the training progress by setting model to inference only mode with *model.eval()*. Output predictions will then be used to calculate both *Lovász-Softmax loss* and *mIoU* as performance metrics. The training loop used *Lovász-Softmax loss* to check for improvement. If there is no improvement for over *patience* continuously epochs, the training loop will be terminated to save time during hyper-parameter tuning. I specified *patience* parameter manually at 20 epochs.

4. Regularization

I performed 2 regularization techniques to avoid over-fitting including image data augmentation and dropout layers insertion.

4.1. Data Augmentation

Since human face inside the input images from *CelebAFaceMask-HQ* dataset are already aligned, using some transformation operation may not make senses. (e.g. Vertical Flip, Rotation)

I choose Albumentations as a library for image augmentations. It is simple to use on segmentation mask by apply the same transformation on both image and mask automatically.

These are selected data augmentation operations chosen from *Albumentations* API.

1. *HorizontalFlip*: Randomly flip both input image and segmentation mask horizontally with chance of *0.5*. Face image already aligned, no need for vertical flip or rotation
2. *HueSaturationValue*: Randomly change values of all pixel in the input image in HSV color system.
3. *RandomBrightnessContrast*: Randomly scaling brightness and contrast based on max value of uint8 (255)
4. *Normalize*: Normalize image with specified mean and S.D. values

For *Normalize* operation, I use pixel mean values (*mean*) of (0.5193, 0.4179, 0.3638) and standard deviation (*std*) of (0.2677, 0.2408, 0.2334) which came from input images inspection of both training and validation set. The calculation script located at *./dataset/dataset_mean_std.py*.

There are several parameters that I exposed them for hyper-parameter tuning.

1. *hue_shift_limit*: Specify maximum values to alter *Hue* of input image. Uniformly selected from [0, 45]
2. *sat_shift_limit*: Specify maximum values to alter *Saturation* of input image. Uniformly selected from [0, 64]
3. *val_shift_limit*: Specify maximum values to alter *Value* of input image. Uniformly selected from [0, 64]

Figure 1. WandB Interactive Dashboard

4. *brightness_limit*: Specify maximum values to alter brightness of input image. Uniformly selected from [0.0, 0.25]
5. *contrast_limit*: Specify maximum values to alter contrast of input image. Uniformly selected from [0.0, 0.25]

To be honest, I realized that I've forgot to swap labels of left/right eyes during horizontal flip operation since eye's direction should related to spatial location (right/left eyes should be on the right/left side of the image respectively). I knew this from inspecting instability between left/right eyes in the prediction, which may lower the evaluation result.

4.2. Dropout

Dropout layers also inserted across difference modules after each activation function to prevents neurons from co-adapting too much by randomly set a number of output features of a layer during training to zero. Configurable parameter *dropout_rate* control chance of neuron to be dropped, was selected from *[0.1, 0.2, 0.3]* during hyper-parameter tuning.

5. Hyper-parameters Tuning

I used *Bayesian* optimization method provided by **WandB** which allow me to execute the sweeps process across multiple devices simultaneously. The reason I pick this one over random and grid search is because it help to balance exploration and exploitation with ability to handle continuous, discrete, and categorical parameters. *Bayesian* optimization uses probabilistic models to guide the search for optimal hyper-parameters from defined search space. The algorithm starts by randomly sampling a few configurations. It then builds a probabilistic model of the objective function which in my case is **maximizing *mIoU* of validation dataset**. Hyper-parameters sweeps were conducted through 4 compute instances in parallel for 3 days, result in 57 runs with 52 successful runs and 5 failed from configurations that gave oversized models.

You can access the full report containing all runs from hyper-parameters tuning at an interactive dashboard.

6. Results and Analysis

The best run named *silvery-sweep-3* with ID *bzwql5yt* was picked across all runs from all instances using maximum *mIoU* from validation dataset as criterion. Table 2 show all hyper-parameters used in this run.

Parameter	Value
epochs	128
base_lr	0.0476
scheduler	Cosine
dropout_rate	0.2
artrous_depth	6
contrast_limit	0.0201
hue_shift_limit	28
sat_shift_limit	48
val_shift_limit	17
base_atrous_rate	5
brightness_limit	0.0564
aspp_output_channels	256
backbone_removed_layers	5

Table 2. Best Configuration Parameters

6.1. Results

The best model above use only first 7 layers of *MobileNetV3_small* as backbone module with and *backbone_width_multiplier*=2.34375 (obtained from binary search). The atrous rates used in each atrous convolution branch of the *ConfigurableASPP* module are [5, 10, 15, 20, 25, 30]. It contain **1,997,579 parameters count** and achieved following scores:

1. Validation dataset
   - *mIOU*: **59.5408**
   - *F-measure*: **0.9702**
2. Test dataset
   - *mIOU*: **58.935** (as seen from Figure 2)

6.2. Analysis

Parameters importance in Figure 3 were calculated by using random forest regressor to predict the model's performance (*mIoU* of validation dataset) based on hyper-parameter values. *Parameter correlation*, on the other hand, reveals relationships between different hyper-parameters and their connections to performance metrics, utilizing Spearman's rank correlation coefficient.

This indicate that configurations with relatively high *mIoU* trend to remove a lot of layer from backbone network, allowing it to have enough room for

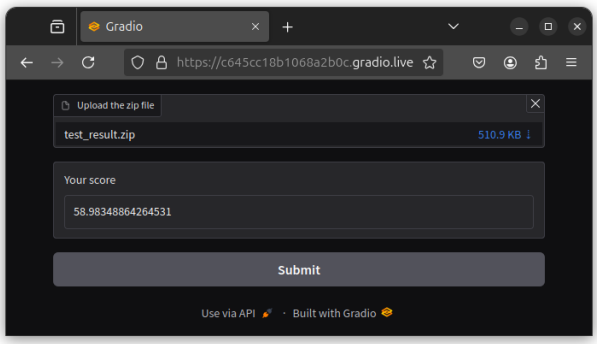


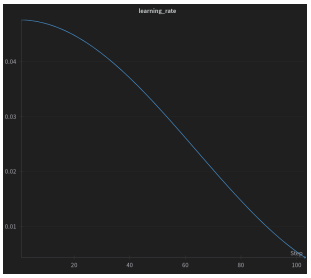
Figure 2. Benchmark Submission Result

Figure 3. Parameters Importance and Correlation

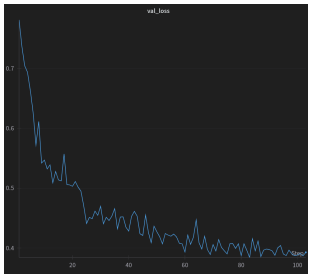
higher count of atrous convolution branches and *backbone_width_multiplier*. *CosineAnnealingLR* scheduler perform better than *LinearLR* while *ReduceLROnPlateau* was failed to ran from my mistake. The best run also use *CosineAnnealingLR* scheduler as seen from learning rate plot in Figure 4a, result in validation loss droping rapidly during initial phase then slightly converge near zero later on illustrated in Figure 4b.

7. Conclusion

The optimized face parsing model, with 1,997,579 parameters, demonstrates strong performance within the given constraints. The best configuration achieves an *mIoU* of **59.5408** and an *F-measure* of **0.9702** on the validation dataset, with an *mIoU* of **58.935** on the test dataset. Key findings from the hyper-parameter tuning process reveal that optimal configurations tend to use fewer layers from the backbone network, allowing for more atrous convolution branches and a higher backbone width multiplier. While the model shows promising results, there's potential for further



(a) Learning Rate plot from best configuration



(b) Validation loss plot from best configuration

Figure 4. Plots from the best configuration

improvement, particularly in handling eye labels swapping during *HorizontalFlip* operation in data augmentation techniques, add more option on scheduler.

References

[1] Maxim Berman, Amal Rannen Triki, and Matthew B. Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks, 2018. 3

[2] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2017. 2

[3] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019. 2

[4] Cheng-Han Lee, Ziwei Liu, Lingyun Wu, and Ping Luo. Maskgan: Towards diverse and interactive facial image manipulation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 1