AI6126 Advanced Computer Vision Project 1

Chen Yongquan (G2002341D)

Nanyang Technological University

Contents

Model

For our model, we used a ResNeSt-50 architecture [1] from PyTorch Hub as the backbone of our network. The network has been pretrained on the ImageNet 2012 dataset. For prediction of the 40 facial attributes, we replace the fully connected layer of the pretrained network with a multi-head binary classifier that gives the prediction for each of the attribute. The ResNeSt-50 architecture is similar to the ResNet-50 architecture and uses 5 convolutional layers. The last 4 layers consists of 3, 4, 6 and 3 bottleneck blocks respectively and each bottleneck block consists of 1x1, 3x3 and 1x1 convolutions sequentially.

ResNeSt improves upon ResNet [2] by using a split-attention block. The split attention block also consists of 1x1, 3x3 followed by 1x1 convolutions but splits convolution into multiple pathways, each pathway and its filters representing a single feature-map group. The number of feature-map groups is a new dimension which the authors of ResNeXt called ‘cardinality’ [3]. The split-attention block further splits each cardinal group into split attention pathways and the number of splits is another dimension called radix. The output from the splits are aggregated using a weighted softmax function while the output from cardinal groups are concatenated channel wise before the final 1x1 convolution. For the pretrained model, a radix of 2 and cardinality of 1 is used.

This is the first main layer of the network which includes a 3 3x3 convolutions followed by a max pooling layer. The first convolution halves the output size (height & width) and outputs 32 filter channels. The third convolution increases channels to 64, and max pooling halves the output size again. This is followed by 4 main layers composed of bottleneck blocks as described previously.

```
(conv1): Sequential(
  (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): ReLU(inplace=True)
  (6): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
(maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
```

The first convolution is a 1x1 convolution followed by the split attention convolution pathways and finally the 1x1 convolution. At the end of each bottleneck block before the final ReLU activation, the input to the block is summed with the intermediary output in order to let the block learn the residual instead. The bottleneck blocks are stacked together in each layer, with the last bottleneck block in each layer doubling the output channels. Layers are then stacked together with each successive layer having double the channels in the previous layer.

```
(0): Bottleneck(
  (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): SplAtConv2d(
    (conv): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=2, bias=False)
    (bn0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (fc1): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1))
    (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (fc2): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))
    (rsoftmax): rSoftMax()
  )
  (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (downsample): Sequential(
    (0): AvgPool2d(kernel_size=1, stride=1, padding=0)
    (1): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

Loss Function

The CelebA dataset is imbalanced with respect to the 40 attributes. For instance, no beard and young are each positive for over 70% of the dataset while bald, mustache and gray hair occurs less than 5%. This can result in overfitting to the frequent attributes and poor performance in the rare ones especially if the rare attributes are also hard to classify and gives a much smaller loss compared to the easy ones. When using cross-entropy loss, the small losses from the hard and rare attributes are then overwhelmed by the larger losses from the easy attributes in majority of the samples, thus the network barely learns to classify any of the hard attributes.

Thus, we use focal loss which can reduce the loss on well-classified attributes in order to focus training on the harder attributes. The formula for focal loss is given by:

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t)$$

γ determines how much the loss is going to focus on poorly classified attributes while α is a weighting factor for each attribute that can offset the class imbalance problem. Focal loss has been shown to improve accuracy in dense object detection for one stage detectors.

To target class imbalance with respect to facial attributes, we use a value of 0.25 for α for attributes with positive samples in more than 25% of the training set and 0.75 for the rare attributes. γ is fixed at 2.0. These hyperparameters gave the best results after training separate models for 20 epochs using a range of values of α, $[0.1, 0.25, 0.5, 1.0]$, and γ, $[5.0, 2.0, 1.0, 0.5]$, then performing validation using the validation set. We also tested using focal loss on data imbalance within each attribute class itself, by using a value of 0.25 for α for positive classes if the either of the binary classes is more than twice the other.

Data Augmentation

To generate more random samples and improve generalization, we perform 3 transforms on the training set.

First, we perform a random crop of the image with scale 0.5 to 1.0 and aspect ratio $3/4$ to $4/3$, then resize the crop to 320 pixels.

Then, we perform a random horizontal flip with probability 0.5.

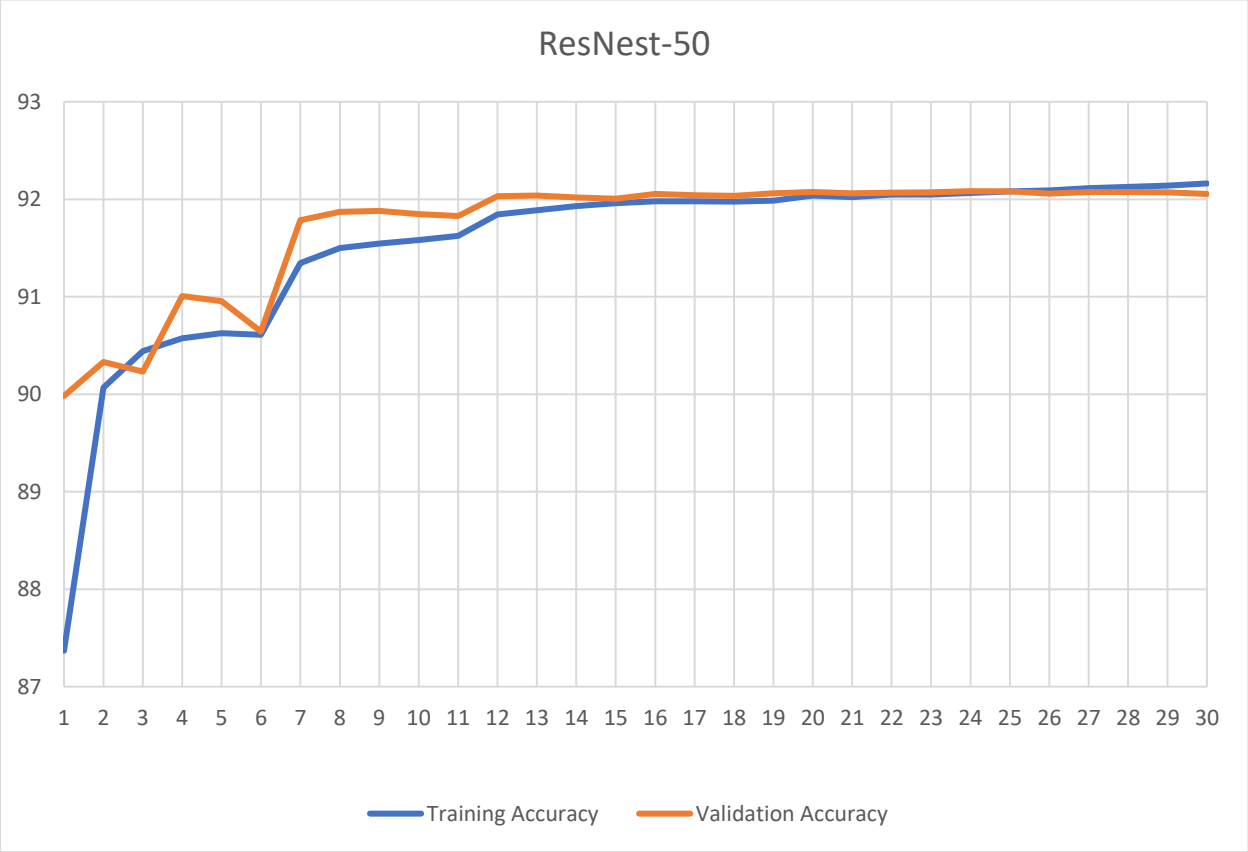
Finally, we jitter the brightness and saturation with a factor of 0.6 to 1.4 and hue from -0.5 to 0.5.

Validation and test images from CelebA dataset are cropped at the center with size 178 pixels then resized to 320 pixels.

Images from the evaluation set are resized to 320 pixels.

Training and Accuracy

Training was done for 30 epochs using a mini-batch size of 32. The training curve is shown below. We used the checkpoint from the epoch with the least difference between training and validation error for evaluation, which is epoch 25. Initial learning rate was 0.1 which was reduced on plateau by a factor of 0.1 with a patience of 1 to a minimum of 0.000001.



Training accuracy at epoch 25 was 92.0796% and loss was 0.0124. Validation accuracy was 92.0807% and loss was 0.0111. Margin between both accuracies was 0.00114% which widened in successive epochs.

Validation Accuracy

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Acc	94.48	86.83	81.92	85.15	99.01	95.85	84.48	84.26	91.4	95.5	96.6	85.17	92.97	95.7	96.77	99.59	96.85	98.14	92.3	88.9
Class	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
Acc	98.87	94.28	96.38	93.76	96.43	76.24	96.38	78.06	94.77	94.59	97.33	93.66	85.58	87.05	92.47	99.09	92.81	88.88	96.08	88.63

Average: 92.0807%

Test Accuracy

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Acc	94.79	84.28	83.45	85.61	99.06	96.11	72.09	85.43	89.79	95.93	96.29	88.48	92.91	95.72	96.35	99.68	97.53	98.2	91.73	87.9
Class	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
Acc	98.37	94.18	97.13	87.87	96.4	76.59	96.66	77.99	93.83	94.67	98.05	93.26	85.26	85.62	91.03	99.05	94.21	87.39	96.5	88.82

Average: 91.60525%

References

[1] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, Z. Zhang, H. Lin, Y. Sun, T. He, J. Mueller, R. Manmatha, M. Li and A. Smola, "ResNeSt: Split-Attention Networks," 2020. [Online]. Available: https://arxiv.org/abs/2004.08955.

[2] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385.

[3] S. Xie, R. Girshick, P. Dollár, Z. Tu and K. He, "Aggregated Residual Transformations for Deep Neural Networks," 2016. [Online]. Available: https://arxiv.org/abs/1611.05431.