Programing Assignement 1: Resnet

Bingnan Li
2020533092
libn@shanghaitech.edu.cn

1. Resnet on CIFAR10

In this section, I will introduce the implementation details and comparison between training from scratch and training with fine-tuning.

1.1. Network Architecture

For this classification task, I use Resnet34 as the backbone and modified two layers to fit the input image size and the number of classes. The detailed modifications are as follows:

- Change the first convolution layer from 7×7 kernel with stride 2 to 3×3 kernel with stride 1.
- Change the final fully-connected layer from 1000 output to 10 output.

1.2. Global Training Settings

1.2.1 Dataset split

The original CIFAR10 dataset contains 60000 images with size 32×32 with 50000 for training and 10000 for testing.

Given that it doesn't naturally split the validation set, I randomly choose 5000 images from the training set as the validation set with a fixed seed.

1.2.2 Data Augmentation

In order to increase the robustness and generalization of the model, I use random horizontal flip and random crop with zero padding as augmentation methods.

1.3. Training from scratch

1.3.1 Training details

The hyperparameters settings follow the detail of [2], which are listed as follows:

- learning rate: 0.1
- learning rate decay: decay 0.1 after 32k and 48k iterations.

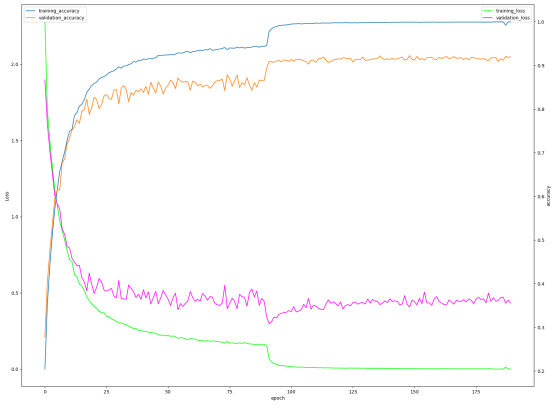

Figure 1. Training curve of Resnet34 on CIFAR10

- batch size: 128

Moreover, I used *Cross Entropy loss* as loss function and used classification accuracy as metric

1.3.2 Experimental results

After 200 epoch training, the average classification accuracy is **91.31**% under test set. The training curve are shown as Figure1.

1.4. Training from Fine-tuning

Since I used *Resnet*34 as backbone, I can directly load the pretrained weight of *Resnet*34 trained on *ImageNet*.

1.4.1 Training Policy

Given that I only changed the first and the last of layer of *Resnet*34, I choose *semi-freezing training* policy to fine-tune.

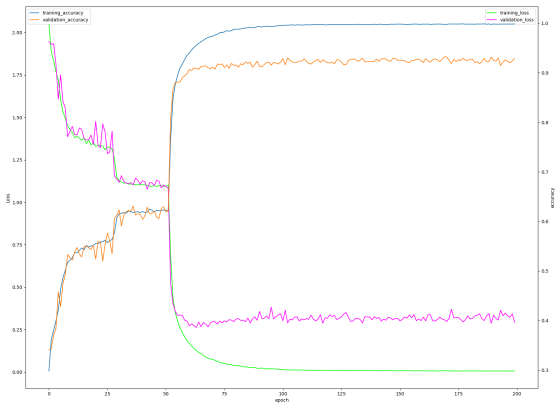

Figure 2. Training curve of Resnet34 on CIFAR10 with fine-tuning

Table 1. Comparison between different methods

method	accuracy	epoch to converge
from scratch	91.31%	100
fine-tuning	**91.67**%	**75**

In detail, I freeze all the middle layers except the first and the last layer, and only train those two layers with learning rate 0.01 for the first 50 epoch. Then I unfreeze all the layers and train the whole network with for the next 150 epoch. The learning rate decay changed to decay 0.1 after 10k and 35k iterations.

1.4.2 experimental results

After 200 epoch training, the average classification accuracy is **91.67**% under test set. The training curve is shown as Figure2.

1.5. Results Comparison and Analysis

I compared the accuracy and the epoch needed to converge[1] between training from scratch and training with fine-tuning in Table 1.

From the table, we can see that the accuracy of training with fine-tuning is slightly higher than training from scratch. Moreover, the training with fine-tuning converges faster than training from scratch.

[1]The epoch needed to converge is defined as the epoch after which the training loss doesn't decrese significantly.

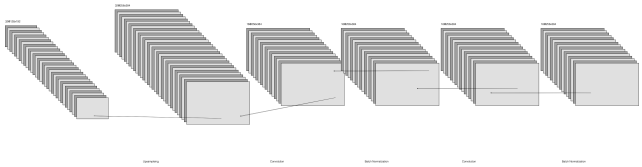

Figure 3. Up-sampling block

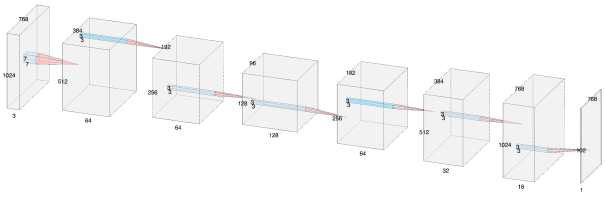

Figure 4. Network architecture of Resnet18 on Crowd Counting

2. Resnet on Crowd Counting

In this section, I explored the capability of *Resnet* on Crowd Counting task. I used *Resnet*18 as backbone and choose *ShanghaiTech Crowd Counting part B* dataset since it has the same image size and relative smaller number of heads.

2.1. Network Architecture

In this task, I followed the heatmap method proposed by [4], so the output of the network is a heatmap with the same size as the input image. To guarantee the output heatmap has the same size as the input image, I performed the following modifications:

- Removed the third and fourth residual block and added three to up-sampling blocks which contains has the architecture of Figure 3.
- Removed the last fully-connected layer

The whole network architecture is shown as Figure 4.

2.2. Training Settings

2.2.1 Dataset split

The original ShanghaiTech dataset contains 716 images with size 768×1024 with 400 for training and 316 for testing. I randomly choose 50 images from the training set as the validation set with a fixed seed.

Table 2. Comparison between different methods

method	MAE	MSE
ours	69.16	103.51
Zhang's [4]	**26.4**	**41.3**

2.2.2 Label preprocessing

The original label is a 2d vector which contains the coordinates of annotated head centers. In order to get the heat map from those labels, I followed the way of [4] to generate the heat map using Gaussian kernel:

$$F(\mathbf{x}) = \sum_{i=1}^{N} \delta(\mathbf{x} - \mathbf{x_i}) * G_{\sigma_i}(\mathbf{x}) \tag{1}$$

where $\sigma_i = \beta \times distance_{avg\_knn}$ is proportional to the average distance of the k nearest neighbors of the ith point. Moreover, following [1,3], I dilated the intensity of heatmap by a factor of 100 so that the model will converge faster.

2.2.3 Loss function

Instead of using the loss function proposed by [4], I used MSE loss as the loss function since I discovered that Zhang's loss function will cause extremly large loss value at the begining when the model is not pretrained from a good initialization.

2.3. Experimental results

Even though I used several tricks to boost training, the final result is still not as good as the result reported by [4]. The comparison is shown in Table 2.

2.4. Analysis

I analyzed the middle layers of the network and found that the model pays more attention to the counter of different semantic regions instead of the heads. I assume there three reasons:

- Resnet18 is not deep enough to capture the fine-grained details of the heads because I visualized the predicted heatmap and discovered that the model pays more attention to the counter of different semantic regions instead of the heads. In Figure 5, you can still see the counter of the bricks.
- The loss function does not constrain the spatial distribution of the predicted heatmap.
- Our output is a heatmap with the same size as the input image. However, in both [4] and [3], the output size is downsampled, which may reduce the difficulty of the task.

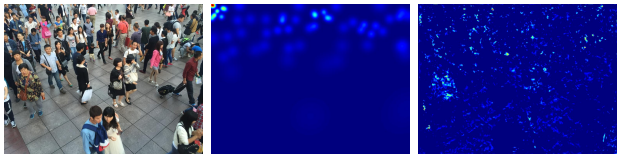

Figure 5. Visualization of the input image, ground truth and predicted heatmap

References

[1] Junyu Gao, Wei Lin, Bin Zhao, Dong Wang, Chenyu Gao, and Jun Wen. C^3 framework: An open-source pytorch code for crowd counting. arXiv preprint arXiv:1907.02724, 2019. 3

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016. 1

[3] Qi Wang, Junyu Gao, Wei Lin, and Yuan Yuan. Learning from synthetic data for crowd counting in the wild. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019. 3

[4] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 589–597, 2016. 2, 3