ECE371 Neural Networks and Deep Learning Assignment 1

Zhenyu Cai 22308003
School of Electronics and Communication Engineering
Sun Yat-sen University, Shenzhen Campus
caizhy28@mail2.sysu.edu.cn

Abstract: In many cases, we are not capable of collecting enourmous amount of data and training a model from the beginning. In such case, adopting existing model and train it on one's own dataset can be the best solution. In this assignment, we use the labeled flower dataset to fine-tune resnet-18 model, and got 93.3% accuracy in a random-split validation set.

Keywords: image classification, fine-tune, resnet-18

1 Introduction

1.1 Tasks

According to the logic and code structure from given main.py, following tasks are required to be done:

(1) Find several methods to preprocess the original images in order to enhance the model's generalization ability.

(2) Modify the last layer of the model so it could fit the number of classes of our dataset.

(3) Define the loss function and optimizer of the model.

(4) In the training progress, add several lines to correctly update the parameters of the model.

(5) Add the function of saving the best model.

1.2 Direct conclusions

According to these tasks, we can directly draw following conclusions:

(1) We can use methods like rotation, rescaling and conversion to grayscale images to reach goal 1.

(2) There are 5 classes in our dataset, if we can find the input size x_{-1} of the last layer in the original model, this can be done with a $x_{-1} * 5$ fully-connected layer.

(3) Since this is a classification model, using cross-entropy loss is the best. As optimizer, we choose SGD here.

2 Related Work

[1] First proposed the concept of convolutional neural networks. By utilizing convolution kernels, researchers have cut down the parameters of networks while improving accuracy. Convolution also enables network learn more complicated spatial features.

[2] Researchers have implemented a deep residual network, greatly enhancing the accuracy of image classification models and laying solid foundation of more powerful modern neural networks.

3 Method

3.1 Data enhancement

According to torchvision.transform, I used following methods to enhance the input images for better robustness and preventing over-fitting: transforms.RandomResizedCrop(224): Randomly crop the image and resize it to 224*224 pixels.

transforms.RandomHorizontalFlip(): Randomly flip the image horizontally.

transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.1): Randomly change the given feature of image. Here, brightness...= 0.4 means these values can change up to 40% from its original value. The same for hue=0.1

transforms.RandomRotation(10): Randomly rotate images within 10 degrees.

transforms.RandomGrayscale(p=0.2): Randomly convert images to gray-scale in probability 20%.

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]): Normalize every channel of the images. The numbers are the mean and standard deviation of ImageNet dataset. This can help to improve the model's generalization ability.

3.2 Modifying model and training methods

As we know, resnet-18 is originally trained on ImageNet, which has 1000 classes. By using:

num_ftrs = model.fc.in_features

We found that num_ftrs=512. Then, we can modify the size of last layer (model.fc) from (512, 1000) to (512, 5) to fit our dataset with 5 classes of flowers.

Since this is a classification model, its training should use cross-entropy loss, which is a commonly-used approach and no further explanation needed. As for optimizers, here we choose SGD with momentum, helping the model to reach convergence faster and more stable. After this, we are ready to use our prepared dataset to update the parameters of the last layer.

3.3 Backward pass and optimizer update

According to torch.tutorial, the general training progress can be divided into several steps: load input and label to device, calculate the output from model, calculate loss function and pass it backward, use optimizer to step forward. Then cycle above steps until it completes all epoch or get the best model. From these steps and the given code, we should add

loss.backward()

optimizer.step()

In the designated line.

3.4 Save model and plotting

After we have completed the training of the model, we would like to save the best-performed model. By analyzing the given code, we can add

torch.save(model.state_dict(), os.path.join(save_dir, 'best_model.pth'))

to save the model's state dictionary. In pratice, researchers and engineers usually save the state dictionary instead of the whole model, since it provides more complexity and compatibility as well as saving memory.

I personally added some lines for plotting the accuracy and loss during the training and validating progress to have a better visual on how they change. The plots will be given below.

4 Experiments

This section should include data tables, figures/charts, and analytical discussion. You are expected to integrate classroom knowledge with your experimental findings for comprehensive discussion, including critical analysis of any identified issues. This section will largely determine the overall execution quality and corresponding grading of the report.

4.1 Analysis of the dataset

The dataset includes 5 classes. The table below shows their name and the amount of images.

class	daisy	dandelion	rose	sunflower	tulip
number of images	118	112	117	108	117

Table 1: Statistic of classes and images

The amount of images of each class are approximately the same, which matches the common rule of dataset for classification model. We extract 80% images for training, and 20% images for validation.

4.2 Training setting and result

Based on the computation resources I can access, I trained the model in following hyper-parameters:

hyper parameter	value
learning_rate	0.001
momentum	0.9
batch_size	32
num_epochs	50

Table 2: Hyper-parameters

The overall training progress took 6m49s on a single NVIDIA Geforce RTX 3090 GPU. In a certain run, it gets 93.33% accuracy on validation. The following figure shows the history of accuracy and loss during training.

[width=0.5]loss$_a cc.png$

Figure 1: History of loss and accuracy

From figure 1, we could draw following conclusion:

(1) The variation of loss and accuracy matches the common trend of deep learning training progress. Specifically, the loss decreases very rapidly at the begining, and then oscillates around a platform. The accuracy follows similar trend, where it increase rapidly at first, and then fluctuate.

(2) The model with best performance is not the one at last. This might be resulted from the random split of dataset. From the oscillation after epoch 10, we know that the model has already reached convergence on this training dataset.

(3) Even with fluctuation, the accuracy on training and validating are about the same, suggesting the model is not over-fitting.

As required, the model state dictionary is saved to best_model.pth. Since the validation set is randomly divided, the accuracy might not be precisely the value given above.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi:10.1109/5.726791.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 1, June 2016. doi:10.1109/CVPR.2016.90.