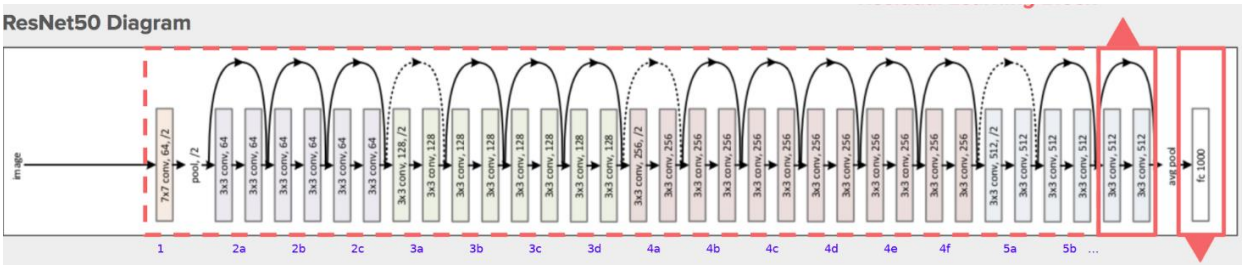
Selected Topics in Visual Recognition using Deep Learning Homework 1 Report

309553013 Chi-An, Lu

1. Introduction

In this assignment, I implemented a CNN model based on ResNet50 to accomplish the in class classification problem on the given car dataset and reached a final accuracy rate about 0.944 on the test dataset. The model is trained for two rounds, 1st round trains 150 epochs and the 2nd for 30 epochs. The Github link: https://github.com/LCA-0907/CS_T0828_HW1

Dataset: 16,185 car images belonging to 196 classes (train: 11,185, test:5000)



Model: fine-tuned ResNet50 with 196 output classes in the last layer

2. Method

a) Model Structure

The whole model is based on ResNet50 structure and pretrained on ImageNet.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

I followed the tutorial at

https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html

modified the last FC layer, change the output from default 1000 to 196.

b) Data-loading

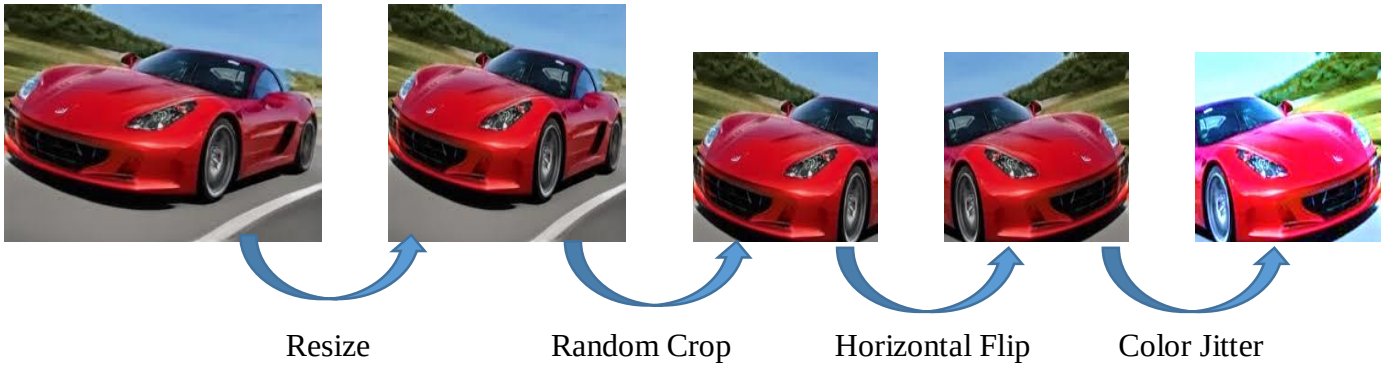
The code first read in training_label.csv file and build the corresponding dictionary for 196 classes. Then write the path to each training data and it's correspond label into a file.

c) Preprocess

For our training and testing data, two customized dataloaders and preprocess function sets are defined.

Training data are processed by the steps followed:

i. Resized into 512*512 pixel
ii. RandomCrop into 448*448 pixel
iii. Randomly horizontal flipped.
iv. Color Jitter(optional)
v. To tensor
vi. Normalize



For testing data:

i. Resized into 512*512 pixel
ii. CenterCrop into 448*448 pixel
iii. To tensor
iv. Normalize

d) Hyper parameters

Batch size = 32

Epochs = 150(120 + 30)

Loss function: cross entropy

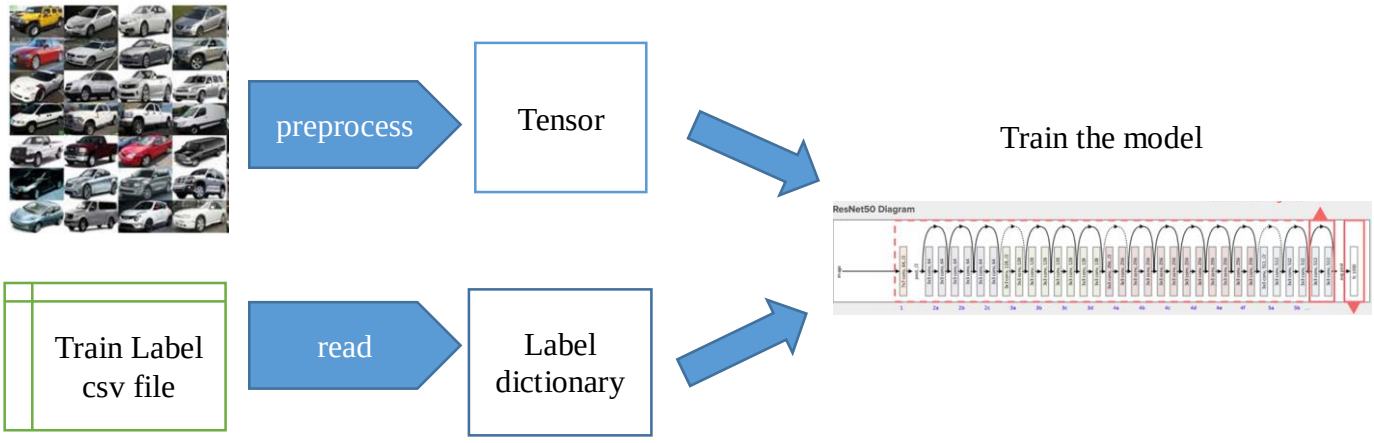
Initial earning rate = 0.01(1st round, 150 epoch), 0.005(2nd round, 30 epoch)

Optimizer: SGD

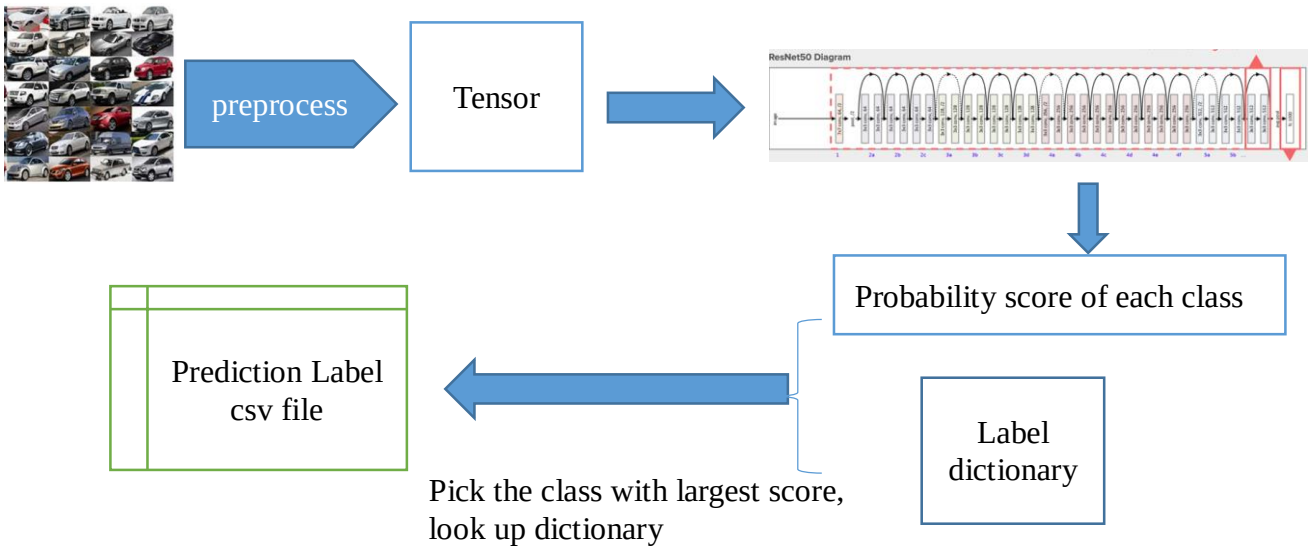
Scheduler: 1step/epoch with γ =0.99

e) Experience Process Flow

Training Phase



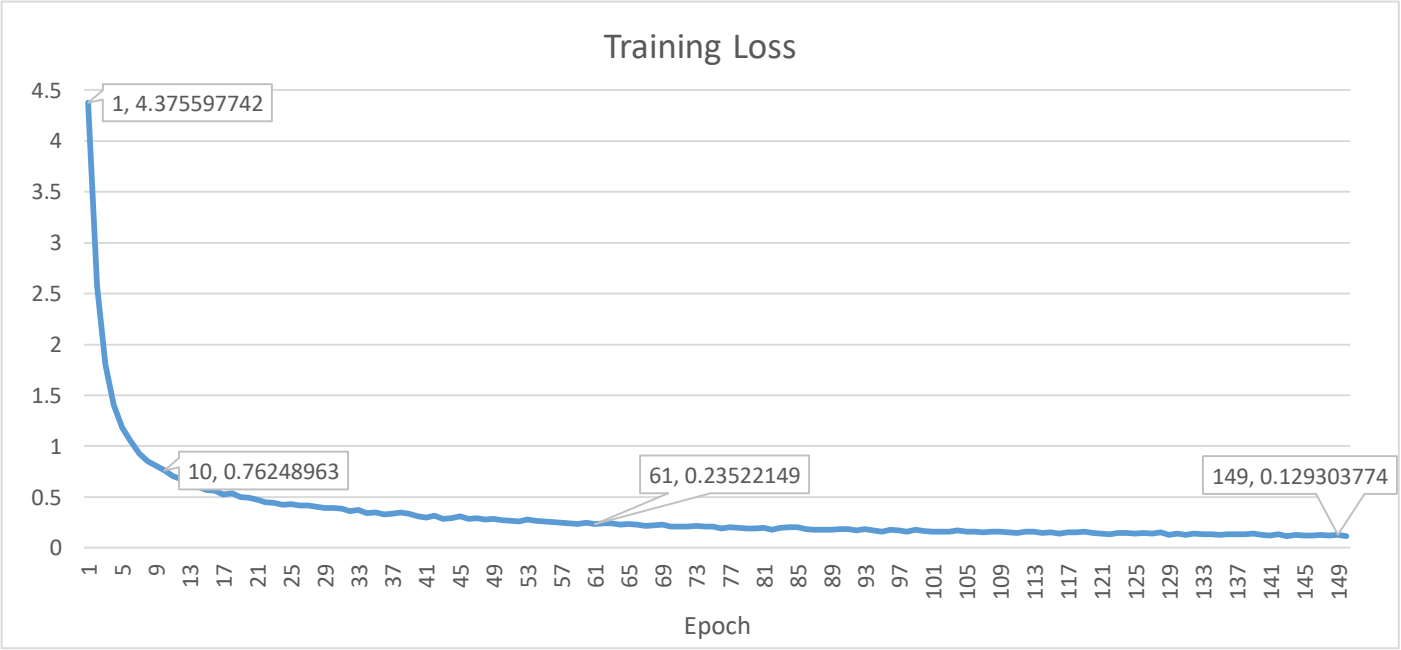
Test Phase



3. Evaluation

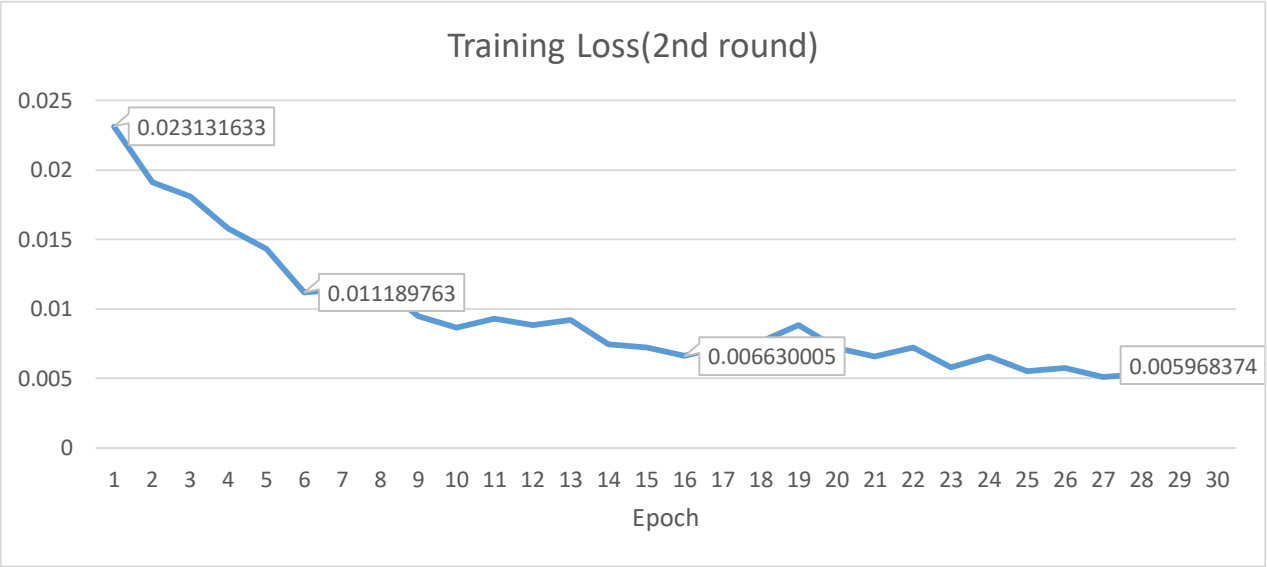
In the first round, train the model for 150 epochs, the evaluated loss is showed in the chart below. The loss descended very slow after 60th epoch.

Testing accuracy scores for epoch 100, 120, 150 are almost the same (about 0.89)



To get better performance, I tried to **retrain the model after Epoch120,** I loaded the saved epoch120 model and adjusted the starting learning rate to 0.05, trained for 30 epochs.

In the 2nd round, I didn't use color jitter function for training image preprocess, below is the training loss. The final model get a testing accuracy score of 0.9454.



4. Summary

ResNet models well performance on classification problems, proper hyperparameter settings and data preprocessing can help a lot in the training.

For training data, since we want it to have more variability, use random crop, horizontal flip, and color jitter to accomplish data augmentation.
During the training phase, use SGD method to update parameters, and the learning rate should be updated in the process, too.In the testing phase, data is resized and center cropped into the same size to training data then feed into the network.

Though ResNet is not the best performanced model on this task nowadays, there are many available tools helping to implement a model base on it. Use the pretrained ResNet can help people wo want to deal with image classification problem achieve the goal.

5. Reference

Finetuning torchvision models(pytorch tutorial):

https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html

Image Augmentation:

https://www.kaggle.com/vishnurapps/image-augmentation-for-more-training-data

Customized Dataloader:

https://www.itread01.com/content/1544541602.html