Assignment 3

Name(s): Ziheng Chen, Yutong Zhang
NetID(s): zihengc2, yutongz7

Part 1: Self-supervised Learning on CIFAR10

1) Rotation training

Report the hyperparameters you used to train your model. Discuss any particular implementation choices which caused significant performance increases.

For our model, we tested various hyperparameters to achieve the best performance. We tried different values for the learning rate and momentum for the optimizer function, as well as different values for the number of epochs, decay epochs, and initial learning rate for training.
For the particular implement choice, we tested the Adam algorithm and SGD algorithm for the optimizer function and found that Adam had a better performance of 78.24%, while SGD had a performance of 79.86%. Therefore, we decided to use SGD for later testing.
As the below tables show, the best performance is when the learning rate of the optimizer function = 0.1, momentum =0.9 for the optimizer function, with 90 epochs, and 15 decay epochs, and initial learning rate = 0.01, which has the best performance of 79.86%.

Optimizer hyperparameters		
Learning rate	Momentum	Performance
0.001	0.9	69.09 %
0.01	0	69.00 %
0.01	0.9	72.38 %

Training hyperparameters			
Number of epochs	Decay epochs	Initial learning rate	Performance
45	15	0.001	72.38 %
90	15	0.001	74.95 %
45	15	0.01	77.92 %

90	**15**	**0.01**	**79.86 %**

2) Fine-tuning late layers

Report the hyperparameters you used to fine-tune your model. Compare the performance between pre-trained model and randomly initialized model.

The performance on record in the table below, based on the data, the performance of the pre-trained model is significantly better than that of the randomly initialized model. The pre-trained model achieves an accuracy of 70.35%, while the randomly initialized model only achieves an accuracy of 44.82%.

Model	Best hyperparameters	Performance
Pre-trained model	num_epochs=20, decay_epochs=10, init_lr=0.01	70.35 %
Randomly initialized model	num_epochs=20, decay_epochs=10, init_lr=0.01	44.82 %

3) Fully supervised learning

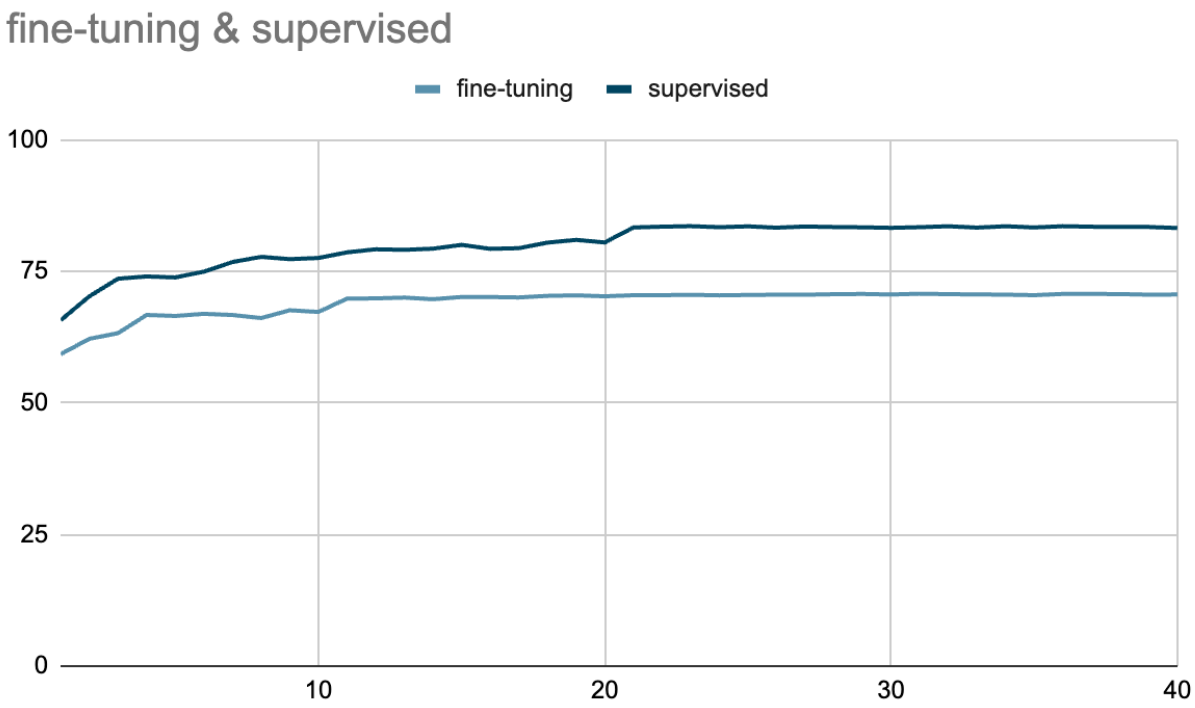
Report the hyperparameters you used to fine-tune your model. Compare the performance between pre-trained model and randomly initialized model. Discuss anything you find interesting comparing fine-tuning the late layers only in section (2) and fine-tuning the whole model in section (3).

The performance on record in the table below, based on the data, the performance of the pre-trained model is significantly better than that of the randomly initialized model. The pre-trained model achieves an accuracy of 84.05%, while the randomly initialized model only achieves an accuracy of 78.62%.

Based on the comparison of the result from sections (2) and (3), the data indicates that fine-tuning the whole model leads to a substantial improvement in accuracy compared to fine-tuning only the late layers. The pre-trained model achieved an accuracy of 70.35% when only the late layers were fine-tuned, whereas the accuracy improved to 84.05% when the whole model was fine-tuned. Similarly, the randomly initialized model had an accuracy of only 44.82% when only the late layers were fine-tuned, but this improved to 78.62% when the whole model was fine-tuned. Also, the difference between the performance of the pre-trained model and the randomly initialized model is much smaller in fully supervised learning (section 3) compared with the performance difference in fine-tuning late layers (section 2).

Model	Best hyperparameters	Performance
Pre-trained model	num_epochs=40, decay_epochs=10, init_lr=0.01	84.05 %
Randomly initialized model	num_epochs=20, decay_epochs=10, init_lr=0.01	78.62 %

4) Extra credit

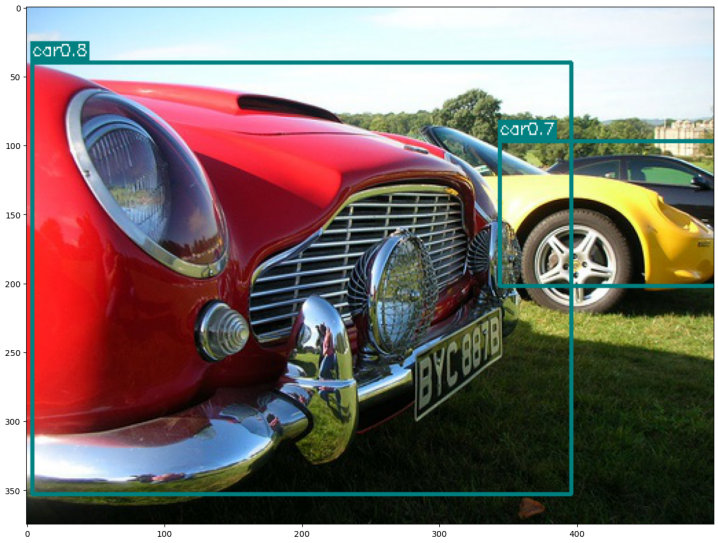


Part-2: Object Detection by YOLO

1. My best mAP value on Kaggle:
    a. 1-mAP on Kaggle: 0.44769
    b. Local test: 0.5521516282664042
2. Did you upload final CSV file on Kaggle: Yes
3. My final loss value :

    a. 'total_loss': tensor(0.8593, device='cuda:0'),
    b. 'reg_loss': tensor(0.6039, device='cuda:0'),
    c. 'containing_obj_loss': tensor(0.0122),
    d. 'no_obj_loss': tensor(0.1600, device='cuda:0'),
    e. 'cls_loss': tensor(0.0832, device='cuda:0')
4. What did not work in my code(if anything): Code mostly worked besides some hiccups. First, I had to drastically reduce the batch_size to 12 to fit my GPU memory. Then there were too many bounding boxes with not high enough confidence, but I found the solution here: https://campuswire.com/c/G333B6F49/feed/613 to fix the problem.
5. Sample Images from my detector from PASCAL VOC:


pottedplant0.71
pottedplar
pottedplan 0.74
pottedplant0.69
pottedplant0.58
pottedplant0.51
chair0.55
pottedplant0.5
pottedplant0.64


car0.8
car0.7