Assignment 3

Name(s): Qi Long, Yihong Yang
NetID(s): qilong2, yihongy3

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.

Num_epochs	50
Decay_epochs	15
Init_lr	0.001
Test Accuracy	78.92%

- Criterion: nn.CrossEntropyLoss()
  Since the task of predicting image rotation degree is a classification task with 4 classes, and the rotation of different degrees are sampled randomly, Cross Entropy Loss is a good loss to use. Compared with MSE loss, Cross Entropy have higher loss at the start of training, which is better for training process.

- Optimizer: optim.Adam(net.parameters(), lr=init_lr, eps=1e-08, weight_decay=0.001)
  Adam optimizer has benefits including adaptive learning rate and regularization. Compared with SGD, Adam makes learning rate picked based on magnitude of gradient and the number of epochs so far, which prevents model from bouncing back and forth without approaching optimal. Besides, implementation of weight_decay=0.001 prevent model from overfitting.

- Adjust Learning Rate: decay every 15 epochs.
  This compulsory shift in learning rate reflects the need of taking smaller updates as model getting closer and closer to optimum. Without this implementation, the model makes smaller and smaller update as training goes on, because learning rate is so high that model is bouncing back and forth around optimum instead of approaching.

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.

Num_epochs	40
Decay_epochs	10
Init_lr	0.001
Pretrained Model Test Accuracy	68.16%
Randomly Init Model Test Accuracy	44.45%

The Test Accuracy of Pretrained Model is better than Randomly Init Model Test Accuracy after finetuning layer 4 and fc for 40 epochs. This can be explained that pretrained model learned about image features by rotation task that the frozen front layers can better capture features of the image.

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).

Num_epochs	50
Decay_epochs	10
Init_lr	0.001
Pretrained Model Test Accuracy	84.35%
Randomly Init Model Test Accuracy	84.47%

- Last Layers VS Whole Model:

  The performance of Supervised Training is better than only finetuning on last layers. This may be because supervised training can update parameters of all layers that it is better fitted to actual task, i.e., classification. Since finetune on last layers has limited update access, its capability is also limited.

- Pretrain VS Random Init:

  The performance of both gets close after long process, but closer observation shows that pretrain model has accuracy improved faster than randomly initialized model in starting epochs (Pretrain 65.70% VS Random Init 51.97% after first epoch), which shows that pretrain task can have better weight initialization for supervised model training.

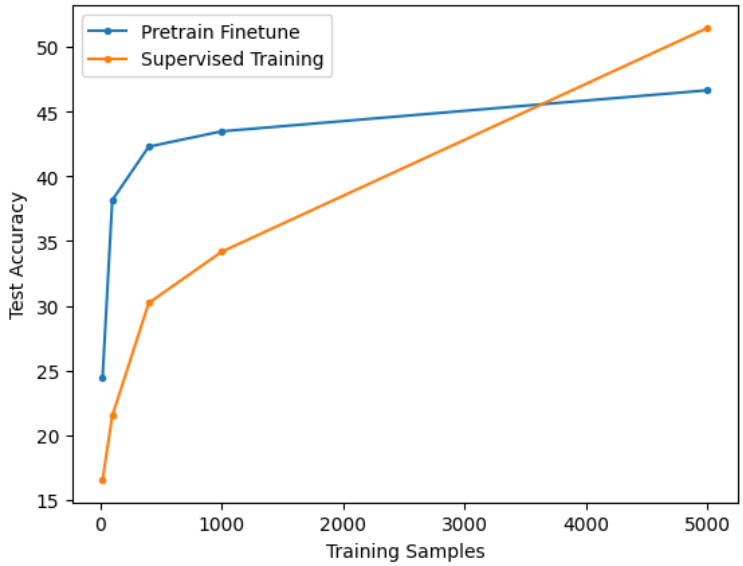
4) Extra credit 1: Accuracy-Sample Plot

Experiment on classification performance vs. number of training examples per category for a supervised CIFAR10 model vs. a RotNet model with the final layers fine-tuned on CIFAR10.

Experiment set up:

- Partial Dataset:
  data_num = [20, 100, 400, 1000, 5000], for each class, sample given number of samples resulting in num*10 samples in total for supervised training or finetuning.
- RotNet:
  Load directly from section 1) (pretrain on whole dataset), freeze all layers except layer 4 and fc, finetune on partial dataset.
- Supervised Model:
  Randomly initialize model, supervised training on partial dataset.

Result Plot:



Discussion:
The result is similar to that in paper. When labeled data is less available, pretrain on unlabeled data using rotation task and only finetune last layers can have better performance than training directly using labeled data. But when the labeled dataset is largely available, supervised training can have better result since it can update parameters of whole model.

5) Extra credit 2: More Advanced Model

Experiment on more advanced model ResNet50 to reach higher rotation and classification task accuracy.

Experiment Setup:
Use the same hyperparameters and implementation details as section 1) and 2). Only change the model from ResNet18 to ResNet50.

Result:

Test Accuracy	Rotation	Classification
ResNet18	78.92%	68.16%
ResNet50	79.33 %	76.13 %

For both tasks, ResNet50 has higher performance than ResNet18.

6) Extra credit 3: Larger Dataset (Based on ResNet18)

Rotation training

Num_epochs	60
Decay_epochs	20
Init_lr	0.001
Test Accuracy	59.75%

- Criterion: nn.CrossEntropyLoss()
  From our previous conclusion, the task of predicting image rotation degree is a classification task with 4 classes, and the rotation of different degrees are sampled randomly, Cross Entropy Loss is a good loss to use.

- Optimizer: optim.Adam(net.parameters(), lr=init_lr, eps=1e-08, weight_decay=0.001)
  Similarly, adam optimizer has benefits including adaptive learning rate and regularization. Compared with SGD, Adam makes learning rate picked based on magnitude of gradient and the number of epochs so far, which prevents model from bouncing back and forth without approaching optimal. Besides, implementation of weight_decay=0.001 prevent model from overfitting.

- Adjust Learning Rate: decay every 10 epochs.
  This compulsory shift in learning rate reflects the need of taking smaller updates as model getting closer and closer to optimum. Without this implementation, the model makes smaller and smaller update as training goes on, because learning rate is so high that model is bouncing back and forth around optimum instead of approaching.

Fine-tuning late layers

Unfreeze layer 4 and fc based on previous training results.

Fully supervised learning

Num_epochs	40
Decay_epochs	20
Init_lr	0.001
Test Accuracy	75.69%

Part-2: Object Detection by YOLO

1. My best mAP value on Kaggle : 0.49987
2. Did you upload final CSV file on Kaggle: Team-Qi Long 25
3. My final loss value : 1.751
4. What did not work in my code(if anything):
5. Sample Images from my detector from PASCAL VOC:

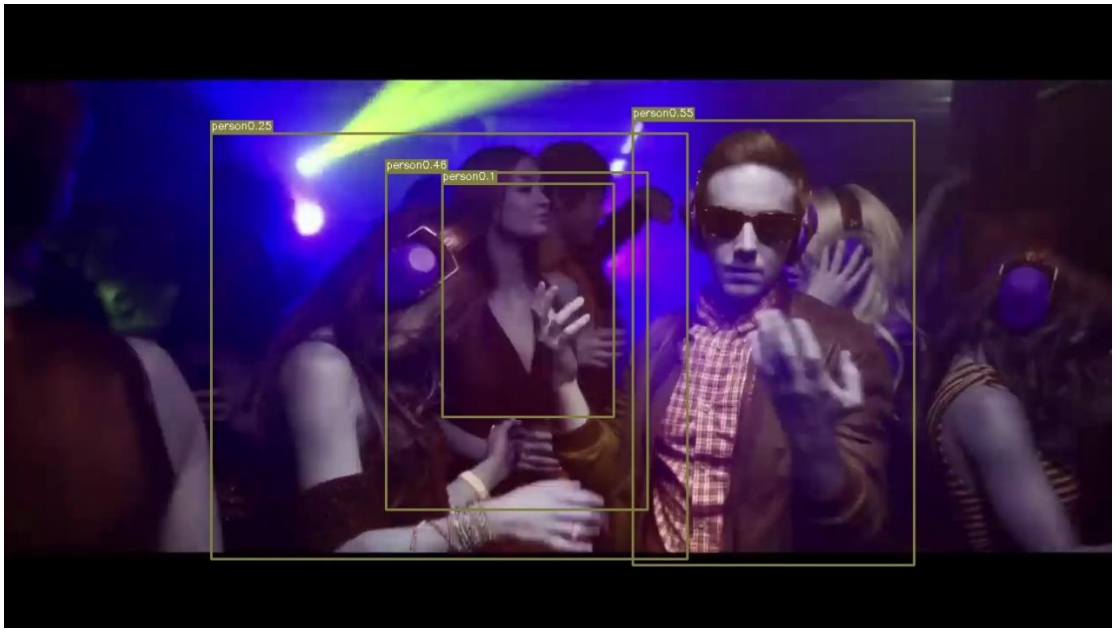
Extra Credit for YOLO:

Extra Credit 1: Video Detection

Experiment on YouTube Video -- SNL Digital Short: YOLO – SNL

Implementation:

1) Video-to-Frame: cutting video clip to 2632 frames (time interval = 2).
2) Object Detection: best resnet50 trained model, example of output frame.



3) Frame-to-Video: fps = 24, use cv2.VideoWriter to transform frame back to video.

Result: video is attached in submission.

Extra Credit 2: Try Another Pre-trained Network

Experiment with ResNet101.

Implementation:
Modify resnet_yolo.py file to load ResNet101 pretrained model.
Hyperparameters: same as ResNet50

B (number of bounding boxes per cell)	2
S (width/height of network output grid)	14
Num_epochs	50
Batch_size	24

Init_lr	0.001
Lambda_coord	5
Lambda_noobj	0.5

Result:

Train Loss 1.538

Test MAP 0.52478

Sample test output: