Spring 2024 CS 444

Assignment 3: Self-supervised and transfer learning, object detection

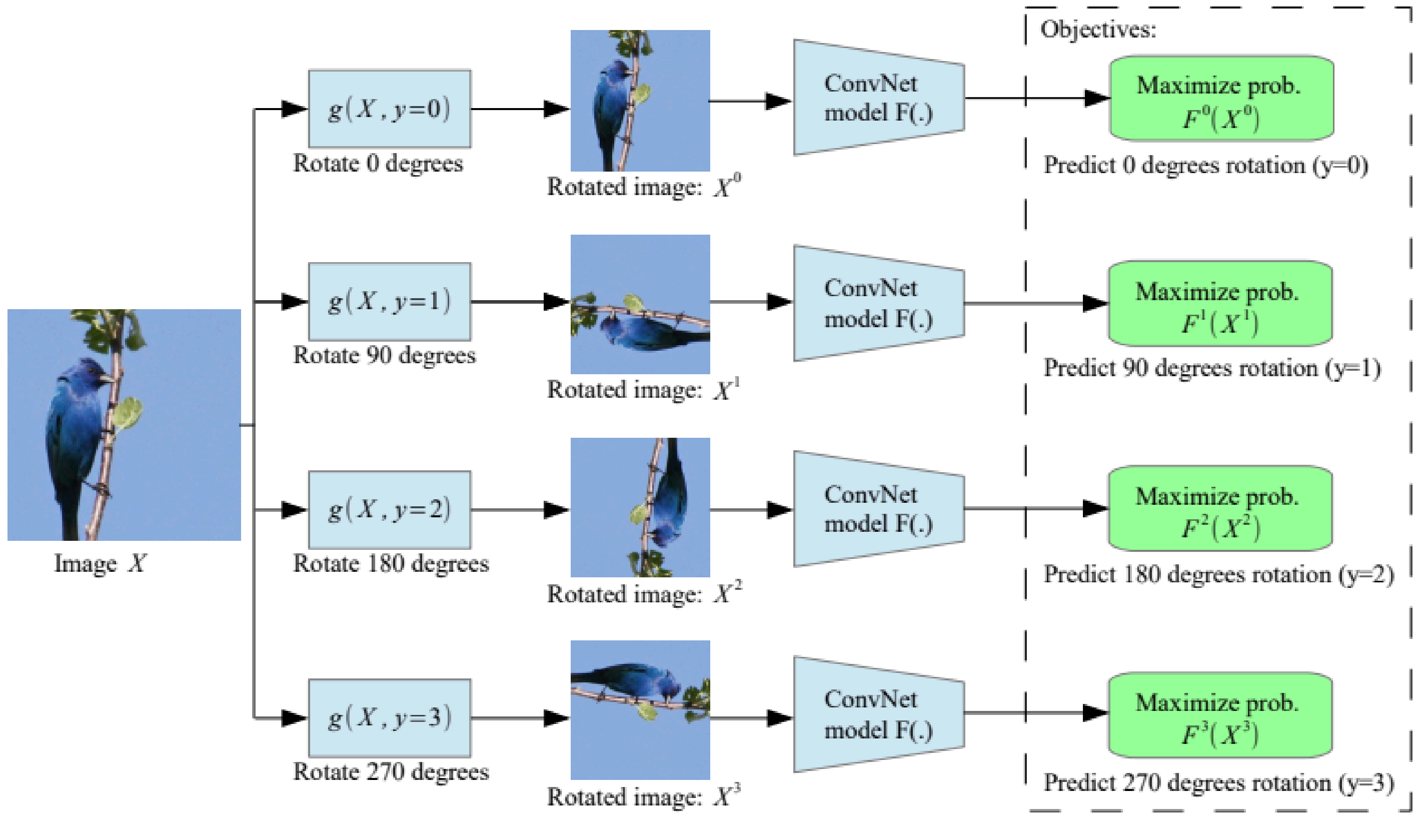
Due date: Wednesday, April 3, 11:59:59 PM

This assignment consists of two substantial and disjoint parts whose goals are to help you gain experience with PyTorch, learn how to use pre-trained models provided by the deep learning community, and adapt these models to new tasks and losses. In Part 1, you will use a simple self-supervised rotation prediction task to pre-train a feature representation on the CIFAR10 dataset without using class labels, and then fine-tune this representation for CIFAR10 classification. In Part 2, you will complete the implementation of an object detector based on YOLO and train it on the PASCAL dataset. Both parts will use PyTorch and Part 2 is likely to require the use of Google Colab Pro or Google Cloud Platform (GCP).

Starter Code

Download the starter code for both parts **here** -- and see setup instructions below under **Assignment Setup**.

Part 1: Self-Supervised Learning by Rotation Prediction on CIFAR10



Source: Gidaris et al. (2018)

In Part 1, you will use PyTorch to train a model on a self-supervised task, fine-tune a subset of the model's weights, and train a model in a fully supervised setting with different weight initializations. You will be using the CIFAR10 dataset, which is a dataset of small (32x32) images belonging to 10 different object classes. For self-supervised training, you will ignore the provided labels; however, you will use the class labels for fine-tuning and fully supervised training.

The model architecture you will use is ResNet18. We will use the PyTorch ResNet18 implementation, so you do **not** need to create it from scratch.

The self-supervised training task is image rotation prediction, as proposed by Gidaris et al. in 2018. For this task, all training images are randomly rotated by 0, 90, 180, or 270 degrees. The network is then trained to classify the rotation of each input image using cross-entropy loss by treating each of the 4 possible rotations as a class. This task can be treated as pre-training, and the pre-trained weights can then be fine-tuned on the supervised CIFAR10 classification task.

The top-level notebook (`a3_part1_rotation.ipynb`) will guide you through all the steps of training a ResNet for the rotation task and fine-tuning on the classification task. You will implement the data loader, training and fine-tuning steps in Pytorch based on the starter code. In detail, you will complete the following tasks:

1. Train a ResNet18 on the Rotation task. Based on the CIFAR10 dataloader, you will first generate the rotated images and labels for the rotation task. You will train a ResNet18 on the rotation task, report the test performance and store the model for the fine-tuning tasks. For the test performance, find the lowest test loss and report the corresponding accuracy. **The expected rotation prediction accuracy on the test set should be around 78%**.

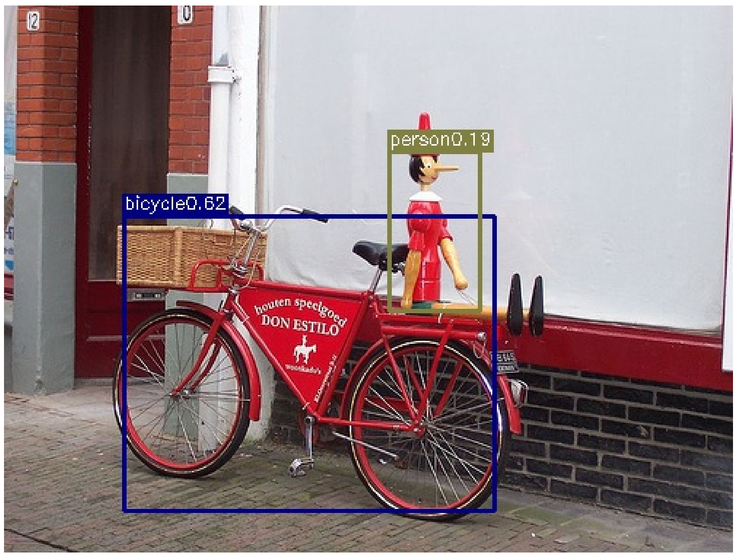
2. Initializing from the Rotation model or from random weights, fine-tune only the weights of the final block of convolutional layers and linear layer on the supervised CIFAR10 classification task. Report the test results and compare the performance of these two models. **The expected test set accuracy for fine-tuning the specified layers of the pre-trained model should be around 60%.**

3. Initializing from the Rotation model or from random weights, train *the full network* on the supervised CIFAR10 classification task. Report the test results and compare the performance of these two models. **The expected test set accuracy for fine-tuning the whole pre-trained model should be around 80%.**

Part 1 Extra Credit

- In Figure 5(b) from the Gidaris et al. paper, the authors show a plot of CIFAR10 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. The plot shows that pre-training on the Rotation task can be advantageous when only a small amount of labeled data is available. Using your RotNet fine-tuning code and supervised CIFAR10 training code from the main assignment, try to create a similar plot by performing supervised fine-tuning/training on only a subset of CIFAR10.
- Use a more advanced model than ResNet18 to try to get higher accuracy on the rotation prediction task, as well as for transfer to supervised CIFAR10 classification.
- If you have a good amount of compute at your disposal, try to train a rotation prediction model on the larger ImageNette dataset (still smaller than ImageNet, though).

Part 2: YOLO Object Detection on PASCAL VOC



In Part 2 you will implement a YOLO-like object detector on the PASCAL VOC 2007 dataset to produce results like in the above image.

The top-level notebook (`MP3_P2.ipynb`) will guide you through all the steps. You will mainly focus on implementing the loss function of YOLO in the `yolo_loss.py` file. You will be provided a pre-trained network structure for the model. The network structure has been inspired by DetNet, however you are not required to understand it. In principle, it can be replaced by a different network architecture and trained from scratch, but to achieve a good accuracy with a minimum of computational expense and tuning, you should stick to the provided one.

As you start this part, you will realize that training is more computationally intensive than what you are used to. In order to get an idea whether your implementation works without waiting a long time for training to converge, here are some typical values to expect:

Epoch	mAP
5	0.2013
10	0.2545
15	0.2749
20	0.2898
25	0.3069
30	0.3355
35	0.3402
40	0.3347
45	0.2588
50	0.3836

Useful Resources

The instructions in the `yolo_loss.py` file should be sufficient to guide you through the assignment, but it will be really helpful to understand the big picture of how YOLO works and how the loss function is defined.

The following resources are useful for understanding YOLO in detail:

- Lecture 10 **(recommended)**
- Original YOLO paper **(recommended)**
- Great post about YOLO on Medium
- Differences between YOLO, YOLOv2 and YOLOv3
- Great explanation of the Yolo Loss function **(recommended)**

Part 2 Extra Credit

- Pick a fun video like this one, run your detector on it (a subset of frames would be OK), and produce a video showing your results.
- Try to replace the provided pre-trained network with a different one and train with the YOLO loss on top to attempt to get better accuracy.

Assignment Setup

Download the starter code here.

To complete this assignment in a reasonable amount of time, you'll need to use a GPU. This can either be your personal GPU, Google Colab or Colab Pro with GPU enabled, or Google Cloud Platform (we will be distributing coupon codes).

Environment Setup

If you will be working on the assignment on a local machine then you will need a python environment set up with the appropriate packages. We suggest that you use Conda to manage python package dependencies (https://conda.io/docs/user-guide/getting-started.html). Unless you have a machine with a GPU, running this assignment on your local machine will be very slow and is not recommended. Please use Google Colab or Google Cloud Platform for this assignment. Instructions on setting up vm instances can be found here. Running Part 1 in Google Colab is fine, but a fully-trained model for Part 2 can take up to 7-8 hours.

We suggest that you use Anaconda to manage python package dependencies (https://www.anaconda.com/download). This guide provides useful information on how to use Conda: https://conda.io/docs/user-guide/getting-started.html. Ensure that IPython is installed (https://ipython.org/install.html). You may then navigate the assignment directory in terminal and start a local IPython server using the `jupyter notebook` command.

Be careful using GOOGLE CLOUD PLATFORM!! Do not use all of your credits!

Data Setup

Once you have downloaded the zip file, go to the assignment3_part2 directory and execute the download_data script provided:

```
sh download_data.sh
```

Submission Instructions:

The assignment deliverables are as follows. If you are working in a pair, only one designated student should make the submission.

1. Upload five files to **Canvas**:
    1. **netid_mp3_part1_output.pdf**: Your IPython notebook for part 1 with output cells converted to **PDF format**
    2. **netid_mp3_part2_output.pdf**: Your IPython notebook for part 2 with output cells converted to **PDF format**
    3. **yolo_loss.py**: Your main YOLO **python** implementation in part 2 for ease of inspection
    4. **netid_mp3_code.zip**: All of your code (python files and ipynb file) **in a single ZIP file**
    5. **netid_mp3_report.pdf**: Your assignment report (using this template) **in PDF format**

2. Upload your YOLO output file to the **Kaggle competition** for the YOLO detector. Note that the value for the Kaggle is (1 - mAP), so you want to get a value *smaller* than the benchmark.

    Don't forget to hit "Submit" after uploading your files, otherwise we will not receive your submission!

Please refer to course policies on collaborations, late submission, etc.