CMU16824Spring18 / Homeworks / hw2-release

Overview

HTTPS https://bitbucket.org/CMU16824Sprir

Last updated 2018-03-08 Access level Read	0 Open PRs	2 Watchers
	3 Branches	0 Forks

Assignment 2: Weakly Supervised Object Localization

- Visual Learning and Recognition (16-824) Spring 2018
- Created By: Senthil Purushwalkam
- TAs: Lerrel Pinto, Senthil Purushwalkam, Nadine Chang and Rohit Girdhar
- Please post questions, if any, on the piazza for HW2.
- Total points: 100

In this assignment, we will learn to train object detectors in the *weakly supervised* setting. For those who don't know what that means - you're going to train object detectors without bounding box annotations!

We will use the PyTorch framework this time to design our models, train and test them. We will also visualize our predictions using two toolboxes Visdom and Tensorboard (yes, again!). In some questions, I will mention which tool you need to use to visualize. In the other questions where I ask you to visualize things, you can use whichever tool you like. By the end of the assignment, you'll probably realise that Visdom and Tensorboard are good for visualizing different things.

We will implement two approaches in this assignment:

1. Oquab, Maxime, et al. "*Is object localization for free?-weakly-supervised learning with convolutional neural networks.*" Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.
2. Bilen, Hakan, and Andrea Vedaldi. "*Weakly supervised deep detection networks.*" Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

We will be implementing sligtly simplified versions of these models to make the assignment less threatening.

You should read these papers first. We will train and test both approaches using the PASCAL VOC 2007 data again. The Pascal VOC dataset comes with bounding box annotations, but we will not be using that for training.

As always, your task in this assignment is to simply fill in the parts of code, as described in this document, perform all experiments, and submit a report with your results and analyses. We want you to stick to the structure of code we provide.

In all the following tasks, coding and analysis, please write a short summary of what you tried, what worked (or didn't), and what you learned, in the report. Write the code into the files as specified. Submit a zip file (`ANDREWID.zip`) with all the code files, and a single `REPORT.pdf` , which should have commands that TAs can run to re-produce your results/visualizations etc. Also mention any collaborators or other sources used for different parts of the assignment.

Software setup

If you are using AWS instance setup using the provided instructions, you should already have most of the requirements installed on your machine. In any case, you would need the following python libraries installed:

Recent activity

1 commit
Pushed to CMU16824Spring18/hw2-release
71c2cf2 fix: import paths first
Senthil Purushwalkam · 2018-03-08

1 commit
Pushed to CMU16824Spring18/hw2-release
869c220 remove keyboard
Senthil Purushwalkam · 2018-03-08

1 commit
Pushed to CMU16824Spring18/hw2-release
3c55368 keyboard
Senthil Purushwalkam · 2018-03-08

1 commit
Pushed to CMU16824Spring18/hw2-release
3029a20 remove keyboard
Senthil Purushwalkam · 2018-03-08

1 commit
Pushed to CMU16824Spring18/hw2-release
e31260d remove keyboard
Senthil Purushwalkam · 2018-03-08

1 commit
Pushed to CMU16824Spring18/hw2-release
7dbe820 remove keyboard
Senthil Purushwalkam · 2018-03-08

1 commit
Pushed to CMU16824Spring18/hw2-release
6524b5a Fix
Senthil Purushwalkam · 2018-03-04

1 commit
Pushed to CMU16824Spring18/hw2-release
22f3544 fix
Senthil Purushwalkam · 2018-03-04

1 commit
Pushed to CMU16824Spring18/hw2-release
5ac0057 Fix
Senthil Purushwalkam · 2018-03-04

1 commit
Pushed to CMU16824Spring18/hw2-release
0b13d8d Delete extra question
Senthil Purushwalkam · 2018-03-04

1 commit
Pushed to CMU16824Spring18/hw2-release

1. PyTorch (0.2+)
2. TensorFlow (1.3+) (yes, tensorflow too)
3. Visdom (check Task 0)
4. Numpy
5. Pillow (PIL)
6. And many tiny dependencies that come with anaconda

cc29847 fix

Senthil Purushwalkam · 2018-03-04

Task 0: Visualization and Understanding the data structures

AWS users should perform this task on the t2.micro instances or your local machine

Visualization using Visdom

In this assignment, we will use two packages for visualization. In Task 0, we will use visdom. You can install visdom using

```
pip install visdom
```

Visdom is really simple to use. Here is a simple example:

```
import visdom
import numpy as np
vis = visdom.Visdom(server='http://address.com',port='8097')
vis.text('Hello, world!')
vis.image(np.ones((3, 10, 10)))
```

You can start a visdom server using:

```
python -m visdom.server -port 8097
```

Data structures for the assignment

The codebases for R-CNN, Fast-RCNN and Faster-RCNN follow a similar structure for organizing and loading data. It is *very* highly likely that you will have to work with these codebases if you work on Computer Vision problems. In this task, we will first try to understand this structure since we will be using the same. Before we try that, we need to setup the code and download the necessary data.

Data setup

1. First, download the code in this repository.

2. Similar to Assignment 1, we first need to download the image dataset and annotations. If you already have the data from the last assignment, you can skip this step. Use the following commands to setup the data, and lets say it is stored at location `$DATA_DIR`.

```
$ # First, cd to a location where you want to store ~0.5GB of data.
$ wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar
$ tar xf VOCtrainval_06-Nov-2007.tar
$ # Also download the test data
$ wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtest_06-Nov-2007.tar && tar xf VO
$ cd VOCdevkit/VOC2007/
$ export DATA_DIR=$(pwd)
```

1. In the main folder of the code provided in this repository, there is an empty directory with the name `data`.
    - In this folder, you need to create a link to `VOCdevkit` in this folder.
    - If you read WSDDN paper [2], you should know that it requires bounding box proposals from Selective Search, Edge Boxes or a similar method. We provide you with this data for the assignment. You need to put these proposals in the data folder too.

```
# You can run these commands to populate the data directory
$ # First, cd to the main code folder
$ # Then cd to the data folder
$ cd data
$ # Create a link to the devkit
$ ln -s <path_to_vocdevkit> VOCdevkit2007
$ # Also download the selective search data
$ wget http://www.cs.cmu.edu/~spurushw/hw2_files/selective_search_data.tar && tar xf selec
```

Now that we have the code and the data, we can try to understand the main data structures. The data is organized in an object which is an instance of the class `imdb`. You can find the definition of this class in `faster_rcnn/datasets/imdb.py`. For each dataset, we usually create a subclass of `imdb` with specific methods which might differ across datasets. For this assignment, we will use the `pascal_voc` subclass defined in `faster_rcnn/datasets/pascal_voc.py`.

It is important to understand these data structures since all the data loading for both training and testing depends heavily on this. You can create an instance of the `pascal_voc` class by doing something like this:

```
# You can try running these
# commands in the python interpreter
>>> import _init_paths
>>> from datasets.factory import get_imdb
>>> imdb = get_imdb('voc_2007_trainval')
```

If you understand it well, you should be able to answer these questions:

Q 0.1: What classes does the image at index 2018 contain?

Q 0.2: What is the filename of the image at index 2018?

We'll try to use the imdb to perform some simple tasks now.

Q 0.3 Use visdom+cv2 to visualize the top 10 bounding box proposals for image at index 2018. You would need to first plot the image and then plot the rectangles for each bounding box proposal.

Q 0.4 Use visdom+cv2 to visualize the ground truth boxes for image at index 2018.

***Hint**: Checkout `vis_detections` in `test.py` for creating the images.*

Task 1: Is object localization free?

A good way to dive into using PyTorch is training a simple classification model on ImageNet. We won't be doing that to save the rainforest (and AWS credits) but you should take a look at the code here. We will be following the same structure.

All the code is in the `free_loc` subfolder. In the code, you need to fill in the parts that say "TODO" (read the questions before you start filling in code). We need to define our model in one of the "TODO" parts. We are going to call this `LocalizerAlexNet`. I've written a skeleton structure in `custom.py`. You can look at the alexnet example of PyTorch. For simplicity and speed, we won't be copying the FC layers to our model. We want the model to look like this:

```
LocalizerAlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace)
    (2): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace)
    (5): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
  )
  (classifier): Sequential(
    (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    (3): ReLU(inplace)
    (4): Conv2d(256, 20, kernel_size=(1, 1), stride=(1, 1))
  )
)
```

Q 1.1 Fill in each of the TODO parts except for the functions `metric1`, `metric2` and `LocalizerAlexNetRobust`. In the report, for each of the TODO, describe the functionality of that part. The output of the above model has some spatial resolution. Make sure you read paper [1] and understand how to go from the output to an image level prediction (max-pool). (Hint: This part will be implemented in `train()` and `validate`.

Q 1.2 What is the output resolution of the model?

Plotting using tensorboard from pytorch

I've included the definition of the `Logger` class in `logger.py`. In order to plot to tensorboard, you need to create an instance of the `Logger` class and use it's methods to plot things. I've also written a function `model_param_histo_summary` that basically plots the histograms of all weight and gradients of weights if you pass the model as input. You can even plot `Variable` s directly (take a look at the definition of `model_param_histo_summary`).

```
logger = Logger(<log_dir>, name='freeloc')
```

When you're logging to Tensorboard, make sure you use good tag names. For example, for all training plots you can use `train/loss`, `train/metric1`, etc and for validation `validation/metric1`, etc. This will create separate tabs in Tensorboard and allow easy comparison across experiments.

Q 1.3 Initialize the model from ImageNet (till the conv5 layer), initialize the rest of layers with xavier initialization and train the model using batchsize=32, learning rate=0.01, epochs=2 (Yes, only 2 epochs for now).

- Use tensorboard to plot the training loss curve
- Use Tensorboard to plot images and the rescaled heatmaps for only the GT classes for 4 batches in every epoch (uniformly spaced in iterations). You don't need to plot gradients on any other quantities at the moment.
- Use Visdom to plot images and the rescaled heatmaps for only the GT classes for 4 batches in every epoch (uniformly spaced in iterations). Also add title to the windows as `<epoch>_<iteration>_<batch_index>_image`, `<epoch>_<iteration>_<batch_index>_heatmap_<class_name>` (basically a unique identifier)

Q 1.4 In the first few iterations, you should observe a steep drop in the loss value. Why does this happen? (Hint: Think about the labels associated with each image).

Q 1.5 We will log two metrics during training to see if our model is improving progressively with iterations. The first metric is a standard metric for multi-label classification. Do you remember what this is? Write the code for this metric in the TODO block for `metric1` (make sure you handle all the boundary cases). The second metric is more tuned to this dataset. `metric1` is to some extent not robust to the issue we identified in Q1.4. So we're going to plot a metric that is not effected by Q1.4. Even though there is a steep drop in loss in the first few iterations `metric2` should remain almost constant. Can you name one such metric? Implement it in the TODO block for `metric2`. (Hint: It is closely related to `metric1`, make any assumptions needed - like thresholds).

We're ready to train now!

Q 1.5 Initialize the model from ImageNet (till the conv5 layer), initialize the rest of layers with xavier initialization and train the model using batchsize=32, learning rate=0.01, epochs=30. Evaluate every 2 epochs. [Expected training time: 45mins-75mins].

- IMPORTANT: FOR ALL EXPERIMENTS FROM HERE - ENSURE THAT THE SAME IMAGES ARE PLOTTED ACROSS EXPERIMENTS BY KEEPING THE SAMPLED BATCHES IN THE SAME ORDER. THIS CAN BE DONE BY FIXING THE RANDOM SEEDS BEFORE CREATING DATALOADERS.
- Use Tensorboard to plot the training loss curve, training `metric1`, training `metric2`
- Use Tensorboard to plot the mean validation `metric1` and mean validation `metric2` for every 2 epochs.
- Use Tensorboard to plot images and the rescaled heatmaps for only the GT classes for 4 batches in every epoch (uniformly spaced in iterations).
- Use Tensorboard to plot the histogram of weights and histogram of gradients of weights for all the layers.
- Use Visdom to plot images and the rescaled heatmaps for only the GT classes for 4 batches in every other epoch (uniformly spaced in iterations) - that is 15*4 batches. Also add title to the windows as `<epoch>_<iteration>_<batch_index>_image`, `<epoch>_<iteration>_<batch_index>_heatmap_<class_name>` (basically a unique identifier).
- At the end of training, use Visdom to plot 20 randomly chosen images and corresponding heatmaps (similar to above) from the validation set.
- Report the training loss, training and validation `metric1` and `metric2` achieved at the end of training (in the report).

Q 1.6 In the heatmap visualizations you observe that there are usually peaks on salient features of the objects but not on the entire objects. How can you fix this in the architecture of the model? (Hint: during training the max-pool operation picks

the most salient location). Implement this new model in `LocalizerAlexNetRobust` and also implement the corresponding `localizer_alexnet_robust()`. Train the model using batchsize=32, learning rate=0.01, epochs=45. Evaluate every 2 epochs.

- For this question only visualize images and heatmaps using Tensorboard at similar intervals as before (ensure that the same images are plotted).
- You don't have to plot the rest of the quantities that you did for previous questions (if you haven't put flags to turn off logging the other quantities, it's okay to log them too - just don't add them to the report).
- In Tensorboard, you can display questions Q1.5 and Q1.6 side by side. This will help you visualize and see if your predictions are improving.
- At the end of training, use Visdom to plot 20 randomly chosen images (same images as Q1.6) and corresponding heatmaps from the validation set.
- Report the training loss, training and validation `metric1` and `metric2` achieved at the end of training (in the report).

Q 1.7 (Extra credit - do this only after Task 2) The outputs of the model from Q1.6 are score maps (or heat maps). Try to come up with a reasonable algorithm to predict a bounding box from the heatmaps.

- Write the code for this in `main.py`.
- Visualize 20 validation images (using anything) with bounding boxes for the ground truth classes (assume that you know which classes exist in the image - plot boxes only for GT classes using the GT labels).
- Note that there is no training involved in this step. Just use the pretrained model from Q1.6.
- Evaluate the mAP on the validation set using the new bounding box predictor that you have created (hopefully you know how to do it using IMDBs). The performance will be bad, but don't worry about it.

Task 2: Weakly Supervised Deep Detection Networks

First, make sure you understand the WSDDN model.

We're going to use an existing PyTorch Faster-RCNN repository to implement WSDDN. So there are many parts of the code that are not relevant to us. Compile the code by doing this:

```
$ cd faster_rcnn
$ #Activate conda pytorch environment
$ conda install pip pyyaml sympy h5py cython numpy scipy
$ pip install easydict
$ ./make.sh
```

(If the above step produces error, report it on piazza).

The main script for training is `train.py`. Read all the comments to understand what each part does. There are three major components that you need to work on:

- The data layer `RoIDataLayer`
- The network architecture and functionality `WSDDN`
- Visualization using both Tensorboard and Visdom

Q2.1 In `RoIDataLayer`, note that we use the `get_weak_minibatch()` function. This was changed from `get_minibatch` used in Faster-RCNN. You need to complete the `get_weak_minibatch` function defined in `faster_rcnn/roi_data_layer.py`.

You can take a look at the `get_minibatch` function in the same file for inspiration. Note that the `labels_blob` here needs to a vector for each image containing 1s for the classes that are present and 0s for the classes that are not.

Q2.2 In `faster_rcnn/wsddn.py`, you need to complete the `__init__`, `forward` and `build_loss` functions.

Q2.3 In `train.py` and `test.py`, there are places for you perform visualization (search for TODO). You need to perform the appropriate visualizations mentioned here:

In `train.py`,

- Plot the loss every 500 iterations using both visdom and tensorboard (don't create new windows in visdom every time).
- Use tensorboard to plot the histogram of weights and histogram of gradients of weights in the model every 2000 iterations

- Use visdom to plot mAP on the *test* set every 5000 iterations. The code for evaluating the model every 5k iterations is not written in `train.py`. You will have to write that part (look at `test.py`)
- Plot the class-wise APs in tensorboard every 5000 iterations
- Again make sure you use appropriate tags for tensorboard

In `test.py`,

- Use tensorboard to plot images with bounding box predictions. Since you evaluate every 5000 iterations during training, this will be plotted automatically during training.

Q2.4 Train the model using the settings provided in `experiments/cfgs/wsddn.yml` for 50000 iterations.

Include all the code, downloaded images from visdom, tensorboard files and screenshots of tensorboard after training. Also download images from tensorboard for the last step and add them to the report. Report the final class-wise AP on the test set and the mAP.