



Assignment #1**Due: Mar. 2nd****Object Classification with PyTorch***Instructor: Abhinav Gupta**TA: Yufei Ye, Nadine Chang, Donglai Xiang***Submission Instructions.**

1. Please submit your solutions to the exercise to the Gradescope. (Entry code: M7Z32E).
2. Your hand-in should consist of two parts: 1) a **single** PDF *report* with your results and analyses, as required; 2) all your *code* in a .zip file (less than 10 MB) which contains all your code files. Please name your report as 'AndrewID.pdf' and code zip as 'AndrewID.zip'.
3. You **should** follow the code structure we defined in the steps of this assignment. Feel free to google how to do certain things if you get stuck. However, it is **not** acceptable to copy paste the whole code repo. For example, it is okay to google “how to add cross entropy loss in pytorch”. But you are now allowed to search “code using alexnet for PASCAL classification” and then clone that code.
4. **Late days:** Everyone have 7 non-penalty late days for the whole semester, which are tracked by Gradescope. After 7 days are used up, any assignment handed in late will be graded as **0** points.
5. We have tested our assignment with best efforts before release. However, with probability 1, there are some typos or even mistakes. If anything does not make sense to you, please let us know!
6. Start Early... Start Early... Start Early...
7. Have fun :p

Introduction to Assignment 1. In this assignment, we will learn to train multi-label image classification models using the [Pytorch](#) framework. We will classify images from the PASCAL 2007 dataset into the objects present in the image.

There are 6 tasks in total. Each involves both coding  and write-ups . In the report, you are expected to include all answers to questions marked with a write-up logo.

Task 0: Fashion MNIST classification in Pytorch (10 points)

The goal of this task is to get familiar with Pytorch, get a flavour of deep learning computer vision and learn to debug your model.

[Fashion MNIST](#) is a dataset of [Zalando's](#) article images — consisting of 70,000 grayscale images in 10 categories. Each example is a 28x28 grayscale image, associated with a label from 10 classes. ‘Fashion-MNIST’ is intended to serve as a direct **drop-in replacement** for the original [MNIST](#) dataset — often used as the “Hello, World” of machine learning programs for computer vision. It shares the same image size and structure of training and testing splits. We will use 60,000 images to train the network and 10,000 images to evaluate how accurately the network learned to classify images.

To get started, we already provide a sample code file (q0_fashion_mnist.py) to get you familiar with the workflow.

0.1 Calculate the dimension and Hyper-parameter search. (2pts)

When running the script directly, you will get an error:

```
RuntimeError: shape '[64, 1]' is invalid for input of size xxx
```

This is a common mistake indicating inconsistent dimension. Change the flat_dim to make the script runnable. What is the number of flat_dim?

Write-up

Coding

Without modifying the code, you should be able to get a reasonable accuracy within 100 iterations just by tuning some hyper-parameter. Which parameters do you modify? (hint: you can complete this task on CPU)

Write-up

0.2 Play with parameters.(3pt)

How many trainable parameters does each layer have?

Write-up

Coding

0.3 Modify the code. (5pt)

Actually, till this point, there is no non-linearity in the SimpleCNN! (Your TA was surprised at the results at the first place.) Your next task is to modify the code to add on non-linear activation layers, and train your model in full scale.

Where did you modify? (at least 2 lines)

Write-up

Coding

Provide some insights on why the results was fairly good even without activation layers. (**3pt**)

Note! In your debugging process, you should print the model and check the layers before searching hyper-parameters.

Write-up

Modify the code to compute train accuracy. Show the loss and accuracy curves and report your accuracy on test set after 5 epochs.

Write-up

Coding

1 Task 1: Simple CNN network for PASCAL multi-label classification (20 points)

Now let's try to recognize some natural images. We provide some starter code (q2_pytorch_pascal.py) for this task. Following steps will guide you through the process.

1.1 Set up the dataset.

Coding

We start by modifying the code to read images from the [PASCAL 2007 dataset](#). The important thing to note is that PASCAL can have multiple objects present in the same image. Hence, this is a **multi-label** classification problem, and will have to be tackled slightly differently.

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 VOCtest_06-Nov-2007.tar
cd VOCdevkit/VOC2007/
```

1.2 Write a dataloader and data gumentation. (5pts)

Dataloader. Complete the function (preload_anno and __getitem__ in voc_dataset.py).

Coding

The first step is to write a data loader which loads this PASCAL data.

Hint1: Refer to the README in VOCdevkit to understand the structure and labeling.

Hint2: As the function docstring says, the function __getitem__ takes as input the index, and output image, label, and weight. The labels should be 1s for each object that is present in the image, and weights should be 1 for each label in the image, except those labeled as ambiguous (use the attribute 'difficult'). All other values should be 0. For simplicity, resize all images to a canonical size.)

Data Augmentation. Modify __getitem__ to augment the data. Please describe what data augmentation you apply.

Coding

Write-up

Hint: Since we are training a model from scratch on this small dataset, it is important to perform some basic data augmentation to avoid overfitting. Add random crops and left-right flips when training, and do a center crop when testing, etc. As for natural images, another common practice is to subtract the mean values of RGB images from ImageNet dataset. The mean values for RGB images are: [123.68, 116.78, 103.94] – sometimes, rescaling to [-1, 1] suffices.

In the following tasks, we will use data in 'trainval' for training and 'test' for testing, since PASCAL is a small dataset easily to overfit.

1.3 Measure Performance (5pts)

To evaluate the trained model, we will use a standard metric for multi-label evaluation - [mean average precision \(mAP\)](#). Please implement the code for evaluating the model with given dataset in function eval_dataset_map in utils.py. You will need to make predictions on the given dataset with the model and call compute_ap to get average precision.

Coding

Please describe how to compute AP for each class(not mAP).

Write-up

1.4 Modify the Fashion MNIST model to be suitable for multi-label classification. (5pts)

Write the code for training and testing for multi-label classification in `02_pascal.py`. We will be using the same model from Fashion MNIST (bad idea, but let's give it a shot). **Coding**

What is the mAP on test set after training 5 epochs? **Write-up**

1.5 Setup visualization (5pts)

TensorBoard is an awesome visualization tool. It was firstly integrated in **TensorFlow** (possibly the most useful tool TensorFlow provides). It can be used to visualize training losses, network weights and other parameters.

To use TensorBoard in Pytorch, there are two options: **TensorBoard in Pytorch** (for Pytorch $\geq 1.1.0$) or **TensorBoardX** – a third party library. Add code in `02_pascal.py` to visualize the testing MAP and training loss in TensorBoard. **Coding**

Show clear screenshots of the learning curves of testing MAP and training loss for 5 epochs (batch size=20, learning rate=0.001). Please evaluate your model to calculate the MAP on the testing dataset every 100 iterations. **Write-up**

2 Task 2: Lets go deeper! CaffeNet for PASCAL classification (20 points)

Note: You are encouraged to modularize and reuse the code from previous task.

As you might have seen, the performance of our simple CNN mode was pretty low for PASCAL. This is expected as PASCAL is much more complex than FASHION MNIST, and we need a much beefier model to handle it.

In this task we will be constructing a variant of the **alexnet** architecture, known as CaffeNet. If you are familiar with Caffe, a prototxt of the network is available [here](#). A visualization of the network is available [here](#)

2.1 Build CaffeNet (5pts)

Here is the exact model we want to build. In this task, `torchvision.models.xxx()` is **NOT** allowed. We use the following operator notation for the architecture:

1. Convolution: A convolution with kernel size k , stride s , output channels n , padding p (You need to convert VALID / SAME to int number by yourself), is represented as $conv(k, s, n, p)$.
2. Max Pooling: A max pool operation with kernel size k , stride s as $max_pool(k, s)$.
3. Fully connected: For n units, $FC(n)$.

ARCHITECTURE:

```
-> image
-> conv(11, 4, 96, 'VALID')
-> relu()
-> max_pool(3, 2)
-> conv(5, 1, 256, 'SAME')
-> relu()
-> max_pool(3, 2)
-> conv(3, 1, 384, 'SAME')
-> relu()
-> conv(3, 1, 384, 'SAME')
-> relu()
```

```
-> conv(3, 1, 256, 'SAME')
-> relu()
-> max_pool(3, 2)
-> flatten()
-> fully_connected(4096)
-> relu()
-> dropout(0.5)
-> fully_connected(4096)
-> relu()
-> dropout(0.5)
-> fully_connected(20)
```

Screenshot your code to define the model.

Write-up

Coding

2.2 Save the model (5pts)

Please add code for saving the model periodically (save at least 5 checkpoints during training for Task 2). For Task 3 and Task 4, you only need to save the model in the end of training. **You will need these models later.**

Coding

2.3 Train and Test (5pts)

Please modify your code to use the following hyperparameter settings. 1) Adam optimizer with learning rate 0.0001. 2) batch size 32.

Show clear screenshots of testing MAP and training loss for 50 epochs. Please evaluate your model to calculate the MAP on the testing dataset every 250 iterations.

Write-up

2.4 Visualizing: Conv-1 filters (5pts)

Extract and compare the conv1 filters, at different stages of the training (at least from 3 different iterations). Show at least 5 filters.

Write-up

Coding

3 Task 3: Even deeper! Resnet18 for PASCAL classification (15 points)

Note: You are even more encouraged to reuse the code!

Hopefully we all got much better accuracy with the deeper model! Since 2012, many other deeper architectures have been proposed, and [ResNet](#) is one of the popular ones. In this task, we attempt to further improve the performance with the “very deep” ResNet-18 architecture.

3.1 Build Resnet-18 (7pts)

Modify the network architecture from Task 2 to implement the Resnet-18 architecture (refer to the original paper). (In this and next task, you are welcome to use `torchvision.models.xxx()`)

Coding

3.2 Train and Test (8pts)

Use the same hyperparameter settings from Task 2, and train the model for 50 epochs. Report the screenshots of 1) training loss, 2) testing MAP curves, 3) learning rate, 4) histograms of gradients and 5) some ($n \geq 3$) examples of training images from TensorBoard in different iterations. **Write-up**

4 Task 4: Standing on the shoulder of the giants: finetuning from ImageNet (15 points)

As we have already seen, deep networks can sometimes be hard to optimize, while other times lead to heavy overfitting on small training sets. Many approaches have been proposed to counter this, eg, [Krahenbuhl et al. \(ICLR'16\)](#) and other works we have seen in un-/self-supervised learning. However, the most effective approach remains pre-training the network on large, well-labeled datasets such as ImageNet. While training on the full ImageNet data is beyond the scope of this assignment, people have already trained many popular/standard models and released them online. In this task, we will initialize the ResNet-18 model from the previous task with pre-trained ImageNet weights, and *finetune* the network for PASCAL classification.

4.1 Load pre-trained model (7pts)

Load the pre-trained weights up to the second last layer, and initialize last weights and biases from scratch.

The model loading mechanism is based on names of the weights. It is easy to load pretrained models for `torchvision.models` but very often, your model might use different names for the weights. Please briefly explain how to load the weights correctly if the names do not match. *hint*: some [source](#)

Write-up**Coding**

4.2 Train and Test (8pts)

Use similar hyperparameter setup as in the scratch case. Show the learning curves (training loss, testing MAP) for 10 epochs. Please evaluate your model to calculate the MAP on the testing dataset every 100 iterations.

Write-up

5 Task 5: Analysis (20 points)

By now we should have a good idea of training networks from scratch or from pre-trained model, and the relative performance in either scenarios. Needless to say, the performance of these models is way stronger than previous non-deep architectures we used until 2012. However, final performance is not the only metric we care about. It is important to get some intuition of what these models are really learning. Lets try some standard techniques.

5.1 Nearest neighbors (7pts)

Pick 3 images from PASCAL test set from different classes, and compute 4 nearest neighbors of those images over the test set. You should use and compare the following feature representations for the nearest neighbors:

1. fc7 features from the ResNet (finetuned from ImageNet)
2. pool5 features from the CaffeNet (trained from scratch)

Show the 3 images you chose and their 3 nearest neighbors for each case.

Write-up**Coding**

5.2 t-SNE visualization of intermediate features (7pts)

We can also visualize how the feature representations specialize for different classes. Take 1000 random images from the test set of PASCAL, and extract caffeNet (scratch) fc7 features from those images. Compute a 2D [t-SNE projection](#) of the features, and plot them with each feature color coded by the GT class of the corresponding image. If multiple objects are active in that image, compute the color as the "mean" color of the different classes active in that image. Legend the graph with the colors for each object class.

Write-up

Coding

5.3 Are some classes harder? (6pts)

Show the per-class performance of your caffeNet (scratch) and ResNet (finetuned) models. Try to explain, by observing examples from the dataset, why some classes are harder or easier than the others (consider the easiest and hardest class). Do some classes see large gains due to pre-training? Can you explain why that might happen?

Write-up

6 Task 6 (Extra Credit): Improve the classification performance (20 points)

Many techniques have been proposed in the literature to improve classification performance for deep networks. In this section, we try to use a recently proposed technique called [mixup](#). The main idea is to augment the training set with linear combinations of images and labels. Read through the paper and modify your model to implement mixup. Report your performance, along with training/test curves, and comparison with baseline in the report.

Acknowledgements

Parts of the starter code are taken from official PyTorch tutorials. Many thanks to the original authors!