Introduction to Deep Learning for Computer Vision
Assignment 2: Simple Linear Classifier I

Due: January 23, 2019

Abstract

This assignment is aimed at creating a pure `Python` pipeline for training a simple linear classifier, without the help of Deep Learning libraries such as PyTorch. For the first part, we will implement the data loading, as well as the loss function computation. It is intended to make you practice what you learned in lectures by implementing them. Large portion of the implementation is already done for you to make it easier.

Instructions are provided first for a reason.
Please read the instructions carefully.

1 Instructions

1.1 Submission package

Within the homework assignment package, there should be a `submission-package.zip`, which contains the directory structure and empty files for you to get started. Please edit the contents within the file for code, and then create a `zip` archive with the file name `submission-package.zip`, and submit it. **Do not use other archive formats such as `rar` or `tar.gz`.**

1.2 Assignment Report

You do not need to submit a report for this assignment.

1.3 Code

All assignments should be in `Python` 3. Codes that fail to run on `Python` 3 will receive 20% deduction on the final score. In other words, do **not** use `Python` 2.7.

For this assignment, you should **not** need to create additional files. Fill in the skeleton files in the submission package. Do **not** change the name of these scripts.

It is **strongly encouraged** to follow `PEP8`. It makes your code much more readable, and less room for mistakes. There are many open source tools available to automatically do this for you.

1.4 Delayed submission

In case you think you will not meet the deadline due to network speed or any other reasons, you can send an email with the `SHA-256` hash of your `.zip` archive first, and then submit your assignment through email later on. This will **not** be considered as a delay.

Delayed submissions are subject to 20% degradation per day. For example, an assignment submitted 1 minute after the deadline will receive 80% of the entire mark, even if it was perfect.

Submission will close 24 hours after the original deadline, as the solution for the assignment will be released to be used for the next assignment.

1.5 Use of open source code

Any library under any type of open source license is allowed for use, given full attribution. This attribution should include the name of the original author, the source from which the code was obtained, and indicate terms of the license. Note that using copyrighted material without an appropriate license is not permitted. Short snippets of code on public websites such as StackOverflow may be used without an explicit license, but proper attribution should be given even in such case. This means that if you embed a snippet into your own code, you should properly cite it through the comments, and also embed the full citation in a LICENSES file. However, if you include a full, unmodified source, which already contains the license within the source file, this is unnecessary. Please note that without proper attribution, *it will be considered plagiarism.*

In addition, as the assignments are intended for you to learn, (1) if the external code implements the core objective of the task, no points will be given; (2) code from other

CSC486B/CSC586B students will count as plagiarism. To be more clear on (1), with the assignment, we will release a `requirements.txt` file, that you can use with `pip` to setup your environment. On top, you are also allowed to use `OpenCV3.X`, which we will not include in `requirements.txt` as `OpenCV` installation depends on your own framework.

2 Preparing the input data

In this part of the assignment, we will focus on the early stages of the pipeline. We will download the dataset, and extract three types of features. Later, we will try all of them out to figure out which is indeed the best feature to use for our task.

2.1 Downloading and preparing CIFAR10 (0 points)

We've already discussed in Lecture 6 about the input pipeline. Please see Lecture 6 Slides for details on how to obtain CIFAR10 data. The loading of the data is already done for you.

2.2 Preparing the raw RGB data (0 points)

RGB data from CIFAR 10 is in the order of Channel, Height, and Width, or CHW ("channels first") for short. For many image processing libraries, we need to swap them so that they are in the form of Height, Width, and Channel order, or HWC ("channels last").

On a side note, for PyTorch, NCHW is the default order, where N is the sample dimension. Therefore, you want to watch out on what you are doing. Anyways, let's stick to NHWC for now, as the libraries that we are going to use for now expect that.

`load_data` function is already provided for you in `utils/cifar10.py`. Please **do** have a look at it before you proceed any further, so that you understand what is going on.

2.3 Extracting color histograms in HSV space (20 points)

In `utils/features.py`, implement the function `extract_h_histogram` that takes a color image, converts it to HSV and uses the Hue value only to create the histogram with 16 bins. Again, the follow the function definitions in the skeleton file.

When extracting the Hue histogram, you **do not have to** use the weighted vote strategy discussed during the lecture. A simple histogram is sufficient.

Hints Converting color spaces are in many cases already implemented in your library. This is also true for the histogram generation. You probably don't need to do all that yourself. In addition, according to how you load the data, that is, which library you use, you will have either `RGB` or `BGR`. Be careful which space you use for conversion.

Useful functions

- `skimage.color.rgb2hsv`
- `numpy.histogram`
- `numpy.linspace`

2.4 Extracting histogram of oriented gradients (10 points)

In `utils/features.py`, implement `extract_hog` according to the function specs written in the skeleton file. Use the function in `skimage.feature.hog`, with default configurations.

Useful functions

- `skimage.feature.hog`

2.5 Pre-processing (10 points)

Implement the `normalize` function according to the function specs in `utils/preprocess.py`. As we will learn later, it is important to keep loaded data in a reasonable range. It is also especially important that you make your data zero mean. Therefore, as a pre-processing step, we will normalize the input data so that it is zero mean and its standard deviation is one.

Useful functions

- `numpy.mean`
- `numpy.std`

3 Implementing linear classifiers

3.1 Predict function: `solution.py/predict` (10 points)

Implementation the predict function according to the function specifications. Once this function is implemented, you can already test your model with the given linear classifier parameters. The skeleton code should already have this testing part done for you so that you can check if your implementation is correct.

If done properly your model should return the following accuracy:

Table 1: Expected test accuracy.

Configuration	Test Accuracy (%)
Histogram of Oriented Gradients (HOG)	45.67
Hue Histogram	22.28

Note that you can run your script with arguments to easily change your configurations. We use the `argparse` library to enable this. See `config.py` for the available options. Note that some option we will implement later. For example, to test the `h_histogram` option, run

```
python solution.py --feature_type h_histogram
```

3.2 Multiclass SVM

3.2.1 Loss function: `utils/linear_svm.py/model_loss` (15+10 points)

Implement the function according to the specs written in the comments. The equations for computing the loss function was given in Lecture 6. It is defined as

$$L = \frac{1}{N}\sum_{i}^{N} L_i = \frac{1}{N}\sum_{i}^{N}\sum_{j\neq y_i} \max(0, s_j - s_{y_i} + 1) \ , \tag{1}$$

If done properly, your loss function value, depending on feature, should be

Table 2: Expected test loss.

Configuration	Test loss
Histogram of Oriented Gradients (HOG)	3.9286
Hue Histogram	6.9279

Note on use of for loops (10 points): To get full points, you should not use for loops. For loops in `Python` are slow and nasty. For both loss functions, you can implement them without a single for loop. To do this, you should make use of Numpy Broadcasting. We learn this in Lecture 7. If you use for loops you'll at maximum receive 15 points for this part.

3.3 Multiclass logistic regression

3.3.1 Loss function: `utils/logistic_regression.py/model_loss` (15+10 points)

Implement the multinomial logistic regression loss from Lecture 6, which is defined as follows:

$$L = \frac{1}{N}\sum_i^N L_i = -\frac{1}{N}\sum_i^N \log \frac{e^{s_{iy_i}}}{\sum_j e^{s_{ij}}} \quad , \tag{2}$$

where i is the sample index. Or equivalently,

$$L = \frac{1}{N}\sum_i^N L_i = -\frac{1}{N}\sum_i^N \log \frac{e^{s_{iy_i}-C_i}}{\sum_j e^{s_{ij}-C_i}} \quad , \tag{3}$$

where $C_i = \max_j s_{ij}$ to ensure numerical stability.

If done properly, your loss function value, depending on feature, should be

Table 3: Expected test loss.

Configuration	Test loss
Histogram of Oriented Gradients (HOG)	1.7487
Hue Histogram	2.1372

Note on use of for loops (10 points): To get full points, you should not use for loops. For loops in `Python` are slow and nasty. For both loss functions, you can implement them without a single for loop. To do this, you should make use of Numpy Broadcasting. We learn this in Lecture 7. If you use for loops you'll at maximum receive 15 points for this part.

Useful functions

- `numpy.max`
- `numpy.sum`
- `numpy.exp`