Ryerson University
CPS843/CP8307
Introduction to Computer Vision, Winter 2016
Assignment 3 - Machine Learning
Due: Saturday, April 9th, 11:59pm

In this assignment, you will be implementing various parts of a sliding window object detector [1]. In particular, you are tasked with implementing a multi-scale face detector.

All programming questions are to be completed in MATLAB. CPS843 students can work in groups of two, if they choose, and CP8307 students are to work individually. For those working in groups, only submit one copy and clearly indicate the group members. Do not share code with those outside your group or use code from the web (other than the code instructed to use). We will be checking for copying and reserve the right to give offenders (both copier and source) a zero on this assignment and pursue academic sanctions.

1 Face recognition (30 points)

In this part, you need to train an SVM to categorize 36×36 pixel images as "face" or "not face", using HOG features. Use the VLFeat library (`http://www.vlfeat.org/`) for both HOG and the SVM.

You are given:

- a directory of cropped grayscale face images, called `cropped_training_images_faces`
- a directory of images without faces, called `images_notfaces`
- a skeleton script called `generate_cropped_notfaces.m`
- a skeleton script called `get_features.m`
- a skeleton script called `train_svm.m`
- a helper script called `report_accuracy.m` (do not edit this file).

You need to do the following:

1. **[5 points]** Using the images in `images_notfaces`, generate a set of cropped, grayscale, non-face images. Use `generate_cropped_notfaces.m` as your starting point. The images should be 36×36, like the face images.
2. **[5 points]** Split your training images into two sets: a training set, and a validation set. A good rule of thumb is to use 80% of your data for training, and 20% for validation.
3. **[5 points]** Generate HOG features for all of your training and validation images. Use `get_features.m` as your starting point. You are free to experiment with the details of your HOG descriptor. A useful resource is the VLFeat tutorial on HOG: `http://www.vlfeat.org/overview/hog.html`

4. **[5 points]** Train an SVM on the features from your training set. Use `train_svm.m` as your starting point. The parameter "`lambda`" will help you control overfitting. A useful resource is the VLFeat tutorial on SVMs: http://www.vlfeat.org/matlab/vl_svmtrain.html. Note: If you test your SVM on the training set features, you should get near perfect accuracy.

5. **[5 points]** Test your SVM on the validation set features. From the SVM's performance at this step, try to refine the parameters you chose in the earlier steps (e.g., the cell size for HOG, and `lambda` for the SVM). Save your final SVM (weights and bias) in a `mat` file called `my_svm.mat`, and include it in your submission.

6. **[5 points]** Write a script called `recog_summary.m`, which prints out a brief summary of your approach (using `fprintf`). Be sure to include your best accuracy on the validation set, and what you did to improve performance.

2 Face detection (30 points)

In this part, you need to create a multi-scale sliding window face detector.

In addition to the files from Part 1, you are given:

- an image called `class.jpg`
- a skeleton script called `detect.m`
- a directory of grayscale test images, called `test_images`
- bounding box annotations for the test images, called `test_images_gt.txt` (do not edit this file)
- a helper script called `look_at_test_images_gt.m`
- a helper script called `evaluate_detections_on_test.m` (do not edit this file)
- a helper script called `VOCap.m` (do not edit this file).

You need to do the following:

1. Get familiar with the test set, and how bounding boxes work, by exploring `look_at_test_images_gt.m`.

2. **[5 points]** Write a single-scale sliding window face detector, using the SVM you trained in Part 1. Use `detect.m` as your starting point. Evaluate your detector by calling `evaluate_detections.m` with the appropriate arguments.

3. **[10 points]** Upgrade your face detector so that it does not make overlapping predictions. This is called *non-maximum suppression*. Detectors typically yield multiple high scores over a region. You want to report the single best bounding box per object. Since only a single bounding box is reported in the ground truth, failure to do so will result in a reduction in the test accuracy score. You may want to inspect the code in `evaluate_detections_on_test.m` to how to calculate area of intersection, and area of union.

4. **[10 points]** Upgrade your face detector so that it makes predictions at multiple scales.

5. **[5 points]** Use your face detector on `class.jpg`.

6. **[5 points]** Write a script called `detect_summary.m`, which prints out a brief summary of your approach (using `fprintf`). Be sure to include your best accuracy on the test set, what you did to improve performance, and a brief qualitative evaluation of your performance on `class.jpg`.

Bonus points will be awarded to the top-performing classifiers on `class.jpg`. Secret ground-truth labels (bounding boxes for faces) have already been generated. **Do not** use `class.jpg` in anyway to train your detector. If you would like to compete for these points, include a single script called `detect_class_faces.m` which runs the full detection pipeline. This script should:

1. load your SVM from a saved file (`my_svm.mat`),

2. generate features from the image at multiple scales,
3. classify the features,
4. suppress overlapping detections,
5. generate a $N \times 4$ matrix of bounding boxes called `bboxes`, where N is the number of faces you detect,
6. generate a $N \times 1$ variable of SVM scores called `confidences`, and
7. plot the bounding boxes on the image.

The marker will run this script, followed by an evaluation script that will use your `bboxes` and `confidences` to generate an average precision score. The top three performing groups will get points as follows:
1st place: **30 points**
2nd place: **15 points**
3rd place: **10 points**
Feel free to research ways to improve your detector's performance, aside from inputting detections manually. For example, it may help to add new training data to the existing set (e.g., training data from another dataset or augmenting the existing set by applying image transformations to some subset of the images, such as left-right flipping), revise your training approach (e.g., use *hard negative mining*[1]) or add some colour cues to the feature vector. Good luck!

References

[1] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 886–893, 2005.

Acknowledgement: This assignment is adapted from one created by Prof. James Hayes (Georgia Tech).

[1] *Hard negative mining* [1] is a training scheme to improve the performance of a detector. It begins with training a detector on a set of positive examples and an initial set of negative ones. Following this initial training stage, negative examples that are incorrectly classified by the initial model are collected to form a set of hard negatives. These hard negatives are added to the negative training set and a new model is trained. This process may be repeated several times.