Homework Assignment 7: Image Recognition

Due Wednesday, December 7, 2022 at 11:59 pm EST

Description

In lectures, we discussed image recognition using two different approaches: (i) window-based recognition, and (ii) CNN based recognition. In the former, we manually extract features from the template images, e.g., using HOG, and then train a discriminative classifier, e.g., SVM. Once the classifier is trained, we do recognition in a new image by sliding a window over the image; for each window location, we extract the feature vector and classify it using the trained classifier. On the other hand, the convolution layers (base) in the CNN learns features from the given training images, then classification, dense, layers (head) are used for the classification.

In this assignment, you are going to implement these two different approaches with the help of libraries and built-in functions for machine learning and CNNs. The input folder has two subfolders; one for each problem. Each subfolder has training and testing images for your assignment.

What to submit

Download and unzip the `ps7` folder: ps7.zip
Rename it to `ps7_xxxx_LastName_FirstName` (i.e. `ps7_matlab_LastName_FirstName`, or `ps7_python_LastName_FirstName`) and add in your solutions:

`ps7_xxxx_LastName_FirstName/`

- input/ - input images, videos or other data supplied with the problem set
- output/ - directory containing output images and other files your code generates
- ps7.m or ps7.py - code for completing each part, esp. function calls; all functions themselves must be defined in individual function files with filename same as function name, as indicated
- *.m or *.py - Matlab function files (one function per file), Python modules, any utility code
- ps7_report.pdf - a PDF file with all output images and text responses

Zip it as `ps7_xxxx_LastName_FirstName.zip`, and submit on Canvas. **Do NOT include the input folder in the zip file.**

Guidelines

1. Include all the required images in the report to avoid penalty.
2. Include all the textual responses, outputs and data structure values (if asked) in the report.
3. Make sure you submit the correct (and working) version of the code.
4. Include your name and ID on the report.
5. Comment your code appropriately.
6. Please avoid late submission. Late submission is not acceptable.
7. Plagiarism is prohibited as outlined in the Pitt Guidelines on Academic Integrity.

Questions

1. Window based recognition: HOG and SVM

In this part, you will use HOG and SVM libraries to train a classifier that can detect toy cars in an image. The training set has image strips that contain car or anything else. The testing set has full images and you'll need to use window based recognition, along with your trained classifier, to detect the location of the toy car, if any, in each testing image. The images for this problem are found under 'input/p1'; please look at the training and testing images before you proceed. As you preview the images, you should notice that the file name of car images starts with 1, whereas images that don't have a car are saved in files that start with a 0. These in fact are the labels that we are going to use when training the classifier; use a label of 1 to indicate the existence of a car and 0 otherwise.

Any classifiers needs to be trained on some feature input. We use HOG descriptor as our features. The provided training images are of size 96x32, hence, when we use HOG on them, the length of the resultant feature vector should be 1188.

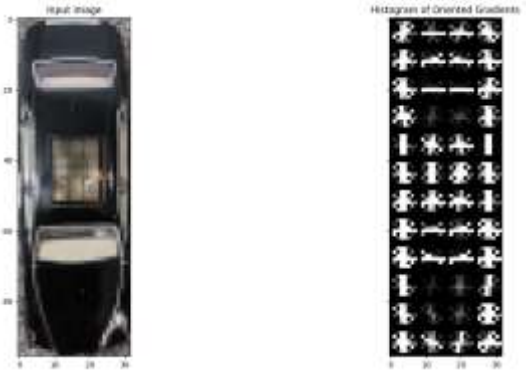
- **In Matlab**, the `extractHOGFeatures` function with the default parameters can do the trick.
- **Python programmers** need to use the hog function from scitkit-image (from skimage.feature import hog). You can leave the default values of the options except for *`cells_per_block`* and *`channel_axis;`* set the former to (2, 2) and the latter to 2. To visualize the hog output (as required in part a below), set *`visualize`* to True.

a. **Trying HOG**: to make sure that you can call the hog descriptor function, compute the HOG descriptor for 'input/p1/car.jpg', and display the feature image. Your output should be similar, but necessarily identical to the image below.

Output: The images showing the input image and the corresponding HOG image as `ps7-1-a.png`

`Output (textual response):`

- The length of the hog feature vector



b. **Training matrix**: Now we should extract features and prepare data for our classifier. To be able to train a classifier, you need to form a training matrix that captures the features extracted from all of your training images and a vector for the corresponding labels (e.g., car [label 1] or not a car [label 0]). Thus, we need to create a matrix X_train and a vector y_train, where **each row** in X_train holds the feature vector extracted from **one** training image. Thus, the number of rows of X_train should equal to the number of training examples, and the number of columns should equal to the number of features extracted per example. Each row in the vector y_train holds the label (0 or 1) of the image whose features are described in the corresponding row in X_train; for example, if the 5th row in X_train has the HOG features from a car training image, then the value of the 5th row in y_train must be 1.
    i. For every image in the training folder 'input/p1/train_imgs', compute the HOG descriptor vector. Append this vector as one row to X_train, and set the corresponding row in y_train with the appropriate label (remember, the first digit of each image name is the corresponding class label; 1 for car and 0 for no car).

Output (`textual response`):
- The dimensions of X_train and y_train.

c. **Training SVM classifier**: Now, having your training data (i.e., X_train and y_train) available, we can train a classifier. In this assignment, we use SVM given its superior performance with small training set. Train a SVM classifier with a linear kernel:
- **MATLAB:** you can use `Mdl = fitcsvm(X_train,Y_train)`
- **Python:** you can use `svm.LinearSVC()` from scikit-learn. You will need to call the fit function and pass your training data and labels.

d. **Detection**: Once you have a trained classifier, you can detect cars in new images. For this part, we will use the sliding window approach. On every image in the testing folder 'input/p1/test_imgs', we would like to detect the location of the toy car, if any. To do so, for every testing image K $(K = 1\ to\ 10)$, use a sliding window of size of 96 (rows) x 32 (columns). For every window position (i, j), i.e., window's top-left corner is positioned at (i, j), compute the HOG feature vector of the window contents, and get the classification score for that feature vector computed at the current window location:
MATLAB: use the function [~, score] = predict(mdl, feature_vector)
Python: call the `decision_function` method of your trained classifier and pass the feature vector you computed from the window.
Hint: the score for a given feature vector will have two columns, only keep the second one and disregard the first. The score is positive if the classifier predict that this might be a car and negative otherwise.

You compute the score for every window location (i, j), for all possible values/locations of (i, j). Since, there's only one toy car in an image, at most, we would look for the location that has the highest score. **If the maximum score is positive and is greater than some**

threshold (say, 1.2), we declare a detection of the car at the location (i_max, j_max) corresponding to that maximum score. Display the testing image K and draw a 96 (height) x 32 (width) rectangle whose top-left corner is at (i_max, j_max). Example output images are below (note, your output might not be identical to these images).

Output: the ten testing images with a box/rectangle showing the location of the detected car, if any, as `ps7-1-d-K.png, where` $K = 1\ to\ 10$

Output (`textual response`):
- Comment on your results.
- Did you get false positives (i.e., your algorithm detected a car where there isn't any car in the image)? False negatives (i.e., missed a car)? What can you do to enhance the performance of your detector?

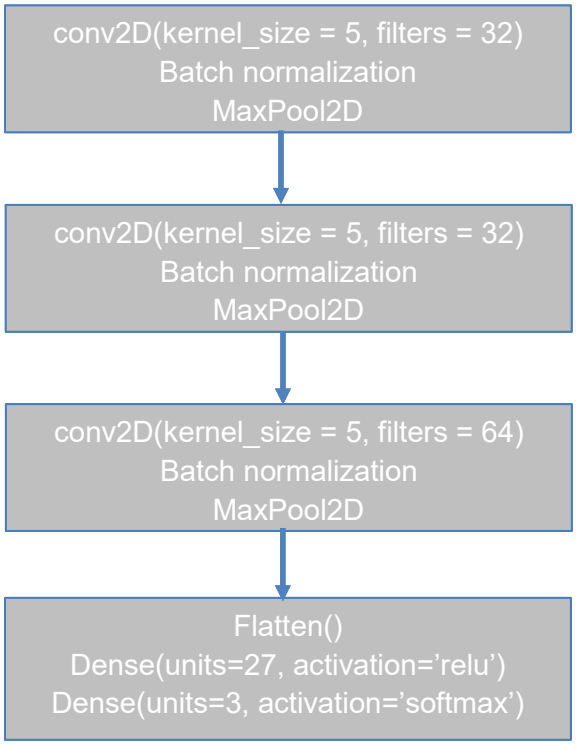
2. Image recognition using convolution neural networks (CNN)

In lectures we described the CNN layers are used to automatically extract features. In this part, we will train a custom CNN model to recognize airplanes, automobiles, and trucks. You're given 32x32 images to train and test your classifier. For each class you have 5000 training images and 1000 testing images.

a. Using the training data ('input/p2/train_imgs'), train a CNN that has the model structure below. Train the network over 10 epochs, using a batch size of 100 images, and zero padding.

Output:
- A screenshot of your model summary `as ps7-2-a-1.png`
- A figure that shows the training accuracy vs epoch as `ps7-2-a-2.png`

Output (`textual response`):
- The accuracy of your trained model on the **training** data.



b. Test the performance of your trained model on the testing dataset ('input/p2/test_imgs').

   Output (`textual response`):
   - The accuracy of your trained model on the **testing** data.
   - How can the testing accuracy be enhanced?

c. For each image in the folder 'input/p2/display_imgs', use your network to predict that image's content (i.e., airplane, automobile, or truck). Display all six images in one figure (as 3x2 subplots) with the predicted labels as the title of each subplot.
   Output: A figure with the six display images along with their network prediction as `ps7-2-c.png`