EE588 Advanced Image Processing
Project #4 Image Classification
(due Monday, Nov. 12, 2018, 10:30:00 am)

Matlab commands are specified in the text and in `blue Courier font` and python commands in parenthesis and in `red Courier font`; variables and filenames (or commands that are consistent between Matlab and python) are specified in black Courier font. In the following python syntax, I have used `import matplotlib.pyplot as plt`, `import skimage.measure`, `from sklearn import svm`, and `import sklearn.metrics`; additionally, you probably want to use `%matplotlib inline` to include figures inline in your jupyter notebook.

Your code should be well commented, and your submission should include enough narration to lead me through your solution. You should be briefly discussing your decisions, your approach to coding, and your results as you progress through the code. Generally this will mean, for each sub-instruction, a brief narrative, your commented code, and the requested output (printed or images displayed). If it is easier, you may upload a separate pdf document for each part (please name each file according to the part so that I can easily navigate—e.g., part_a.pdf). Please also include the last page of this assignment as a cover page to assist me in grading. You may fill that out and upload it as a separate document, or include it as the first page of your merged pdf.

(a) Dataset setup.

In this project, we will use the CalTech101 dataset, which is a standard dataset used for image classification. You can find important information about this dataset at (http://www.vision.caltech.edu/Image_Datasets/Caltech101/). From that webpage, download the dataset itself (http://www.vision.caltech.edu/Image_Datasets/Caltech101/101_ObjectCategories.tar.gz) which is some 131 MB and also the annotations (http://www.vision.caltech.edu/Image_Datasets/Caltech101/Annotations.tar) which will allow us to focus our feature extraction on only the objects in the images. Extract the image dataset and the annotations in your working directory. The images will extract to a `101_ObjectCategories/` directory, under which there are 102 directories named according to the object contained in the image (e.g., `accordion/` or `pizza/`), under which are files with file format `image_XXXX.jpg`, where `XXXX` is a four digit number. The annotations will extract to an `Annotations/` directory, underneath which there are 101 directories named the same categories (for the most part) as the `101_ObjectCategories/` categories, under which are files `annotation_XXXX.mat`, where `XXXX` is a four digit number. There are also 5 other files in the `Annotations/` directory. In order to make subsequent code run more easily:

- Within `101_ObjectCategories/`:
  - Delete directory `BACKGROUND_Google/`
- Within `Annotations/`:
  - Delete `*.mat`
  - Delete `README*`
  - Move (rename) directory `Airplanes_Side_2/` to `airplanes/`
  - Move (rename) directory `Faces_2/` to `Faces/`
  - Move (rename) directory `Faces_3/` to `Faces_easy/`
  - Move (rename) directory `Motorbikes_16/` to `Motorbikes/`

(i) You can store the directory names in a structure (list) with `categories=dir('101_ObjectCategories')` (`categories=sorted(glob.glob('101_ObjectCategories/*'))`). This structure (list) now gives you a means to loop over the 101 different categories of objects in that `categories(k).name` (`categories[k]`) is the `k`-th category name as a string (python will have the string `'101_ObjectCategories/'` prepended to the category name). Using this structure (list), read in the first image (`image_0001.jpg`) from each of the 101 categories and display that image in one location of an 11×10 subplot. Title each of those locations of the subplot with the category name.

(ii) The annotations are stored in Matlab's `.mat` format, which both Matlab and python can load. These annotations can be read in with `ann=load('filename.mat')` (`ann=spio.loadmat('filename.mat')`). This returns a structure (dictionary) with variable names as fields (keys). In this structure `ann.box_coord` (`ann['box_coord']`) is a 1×4 vector of bounding box coordinates and `ann.obj_contour` (`ann['obj_contour']`) is a $2\times K$ vector of pixel locations which outline the contour of the object, where K will be different for different annotations. Read in `Annotations/emu/annotation_0001.mat` and display `box_coord` and `obj_contour`. The object contour points `obj_contour` are (for reasons unbeknownst to me) offset by the `box_coord` coordinates. Read in image `101_ObjectCategories/emu/image_0001.jpg` and display. Plot the annotation outline on top of that with `plot(ann.obj_contour(1,:)+ann.box_coord(3),ann.obj_contour(2,:)+ann.box_coord(1),'w')` (`plt.plot(ann['obj_contour'][0,:]+ann['box_coord'][0,2]-1,ann['obj_contour'][1,:]+ann['box_coord'][0,0]-1,'w')`).

(iii) You can use the object contour outline to define a binary image image mask with `A=poly2mask(ann.obj_contour(1,:)+ann.box_coord(3),ann.obj_contour(2,:)+ann.box_coord(1),M,N)` (`r,c = skimage.draw.polygon(ann['obj_contour'][1,:]+ann['box_coord'][0,0]-1,ann['obj_contour'][0,:]+ann['box_coord'][0,2]-1,(M,N)); A=np.zeros(M,N); A[r,c]=1;` note that the indices are swapped here versus the plot command due to the difference in coordinate systems of image versus plot) where `M`, `N` are the dimensions of the image. Using what you have learned about using structures (lists) to loop over categories, load the first annotation (`annotation_0001.mat`) from each of the 101 categories, use the corresponding `obj_contour` to define an object mask, and display that mask in one location of an 11×10 subplot. Title each of those locations of the subplot with the category name. It might be helpful to read in the image corresponding to the annotation in order to easily get the dimensions. You can use the visualizations in part (a-i) and here to spot-check the correctness of the annotations.

(b) Color features.

(i) Create a function `[f,fnames]=extract_color_features(im,mask)` (`f,fnames=extract_color_features(im,mask)`) with inputs `im`, the image from which to extract features, and the binary annotation mask, `mask`. Outputs will be a length-30 feature vector `f` describing statistics of the colors within the image object and a length-30 cell (list) `fnames` with the feature names. You will extract statistics from the red, green, blue, hue, saturation, and value channels of the image. From each channel, you will compute the mean, standard deviation, median, min, and max value of pixels within the object mask. The command `rgb2hsv` (`skimage.color.rgb2hsv`) may be helpful here. Order your features by channel first in the order given above and by statistic second in the order given above (e.g., your first and second features will be mean and standard deviation of the red channel). Assign brief, descriptive strings for each feature and store those in `fnames` (e.g., `'R_mean'`, and `'R_std'` could be names for the first two features).

(ii) Using `101_ObjectCategories/emu/image_0001.jpg` as the input image `im` and `Annotations/emu/annotation_0001.mat` as the annotation mask `mask`, use your `extract_color_features` function and print out your `f` vector (for conservation of space, please print out as a row vector) and your `fnames` cell (list) (again, please print out horizontally for conservation of space).

(c) Boundary features.

(i) Create a function `[f,fnames]=extract_boundary_features(mask)` (`f,fnames=extract_boundary_features(mask)`) with input `mask`, the binary annotation mask. Outputs will be a length-10 feature vector `f` consisting of the length-10 Fourier descriptors for the object in `mask` and a length-10 cell (list) `fnames` with feature names `'Fourier_desc_a0'`, `'Fourier_desc_a1'`, …, `'Fourier_desc_a9'`. The command `B=bwboundaries(mask)` (`B=skimage.measure.find_contours(mask,0.5)`) will return a cell (list) of arrays (ndarrays) of coordinate pairs associated with the boundary of the object in `mask`. The number of elements in the cell (list) `B` will be the number of contours in the image. The annotated images in this project should have just one contour per image, although you may need to correctly choose the connectivity in the `bwboundaries` (`skimage.measure.find_contours`) function to make that the case. The command `bwboundaries` will return a closed contour, meaning that the last coordinate pair is identical to the first—discard the last coordinate pair in subsequent computations. (If the contour does not intersect the edge of the image, `skimage.measure.find_contours` will close the contour. You can check for this situation by checking whether the first and last coordinate pairs are the same. In such a case, you should discard the last coordinate pair in subsequent computations.) Now you can create the complex signal $s(k)=x(k)+jy(k), k=1,\ldots,K$ where K is the total number of coordinate pairs in the (non-closed) boundary returned by `bwboundaries` (`skimage.measure.find_contours`). Take a length-10,000 FFT of complex signal `s` to yield the length-10,000 Fourier descriptor vector `S`. Bin this length-10,000 Fourier descriptor `S` into a length-10 Fourier descriptor vector `f` using the following code.

```
for k=0:9                                    for k in range(0,10):
 f(k+1) = abs(sum(S(k*1000+1:(k+1)*1000)));    f[k] = np.abs(S[k*1000:(k+1)*1000].sum())
end
```

(ii) Using `101_ObjectCategories/emu/image_0001.jpg` as in the input image `im` and `Annotations/emu/annotation_0001.mat` as the annotation mask `mask`, use your `extract_boundary_features` function and print out your `f` vector (for conservation of space, please print out as a row vector) and your `fnames` cell (list) (again, please print out horizontally for conservation of space).

(d) Region features.

(i) Create a function `[f,fnames]=extract_hu_moments(mask)` (`f,fnames=extract_hu_moments(mask)`) with input `mask`, the binary annotation mask. Outputs will be a length-7 feature vector `f` consisting of the 7 moment invariants (Hu moments—equations 12-39 (11.3-17) through 12-45 (11.3-23)) for the object in `mask` and a length-7 cell (list) `fnames` with feature names `'phi1'`, `'phi2'`, …, `'phi7'`. (Python users may not use skimage.measure.regionprops to compute these moments.)

(ii) Using `101_ObjectCategories/emu/image_0001.jpg` as in the input image `im` and `Annotations/emu/annotation_0001.mat` as the annotation mask `mask`, use your `extract_hu_moments` function and print out your `f` vector (for conservation of space, please print out as a row vector) and your `fnames` cell (list) (again, please print out horizontally for conservation of space).

(iii) The rest of the region features will be gathered from the `regionprops` (`skimage.measure.regionprops`) function. Read through `help regionprops` (`help(skimage.measure.regionprops)`) and you will see that some of the features returned by `regionprops` (`skimage.measure.regionprops`) may not be useful in our image classification situation. For example, the centroid of the object or the orientation of the object may bias the classifier to translation or rotation variance. Using what you learned about the syntax of calling `regionprops` (`skimage.measure.regionprops`), use that function to extract the following 11 features to a feature vector `f` and the names to a cell (list) `fnames`:

- `'Area', 'ConvexArea', 'Eccentricity', 'EquivDiameter', 'EulerNumber', 'Extent', 'FilledArea', 'MajorAxisLength', 'MinorAxisLength', 'Perimeter', 'Solidity'`
- `'area', 'convex_area', 'eccentricity', 'equivalent_diameter', 'euler_number', 'extent', 'filled_area', 'major_axis_length', 'minor_axis_length', 'perimeter', 'solidity'`

(iv) Using `101_ObjectCategories/emu/image_0001.jpg` as in the input image `im` and `Annotations/emu/annotation_0001.mat` as the annotation mask `mask`, print out your `f` vector (for conservation of space, please print out as a row vector) and your `fnames` cell (list) (again, please print out horizontally for conservation of space) for the region features extracted in part (d-iii).

(e) Texture features.

(i) Create a function `[f,fnames]=extract_texture_features(im,mask)` (`f,fnames=extract_texture_features(im,mask)`) with inputs `im`, the image from which to extract features, and the binary annotation mask, `mask`. Outputs will be a length-48 feature vector `f` describing co-occurrence matrix features within the image object and a length-48 cell (list) `fnames` with the feature names.

- Define image `I_q` as the image quantized to 32 levels using `I_q=uint8(round(I*31))` (`I_q=np.round(I*31).astype(int)`), where it is assumed that image `I` is a grayscale image scaled to have values in [0,1]. Furthermore, in order to compute texture measures only over the region mask, set all pixels outside of the mask to 32 and then you can ignore the last row and column in the GLCM.
- Within this function, you will use the command `graycomatrix(I_q,'Offset',offsets,'NumLevels',33,'GrayLimits',[],'Symmetric',true)` , where you will specify `'Offset', [0, d; -d, d; -d, 0; -d, d]` for $d = 1,2,3,4$. (`skimage.feature.greycomatrix(I_q,distances=(1,2,3,4),angles=(0,np.pi/4,np.pi/2,3*np.pi/4),levels=33,symmetric=True,normed=False)`) to compute the gray-level co-occurrence matrix (GLCM) for the set of orientations $\theta = [0,45,90,135]$ degrees and distances $d = [1,2,3,4]$ pixels. Immediately after returning the GLCM matrices, discard the last row and column.
- You can normalize your GLCMs with the following code:

```
for k=1:size(G,3)
    G(:,:,k) = G(:,:,k)/sum(sum(G(:,:,k)));
end
```

```
for d in range(0,4):
    for t in range(0,4):
        G[:,:,d,t]=G[:,:,d,t]/G[:,:,d,t].sum()
```

- For each distance you will have 4 GLCM matrices (one for each orientation). For each distance, and for each of the 6 GLCM features recommended by the `baraldi1995.pdf` paper, you will thus compute 4 different values of each feature (one for each of the orientations). As your ultimate feature value, compute the average and standard deviation across the 4 orientations as recommended by the `haralick1973.pdf` paper.
- Choose feature names that invoke the feature, distance, and whether it is the average or standard deviation across the orientations.

(ii) Using `101_ObjectCategories/emu/image_0001.jpg` as in the input image `im` and `Annotations/emu/annotation_0001.mat` as the annotation mask `mask`, use your `extract_texture_features` function and print out your `f` vector (for conservation of space, please print out as a row vector) and your `fnames` cell (list) (again, please print out horizontally for conservation of space).

(f) Set up feature matrix.

(i) Create code to loop over multiple images, extract features, and build a feature matrix and label vector as follows. Use strings specifying the directories of images from which to extract features, loop over the images in that directory, extract feature vectors `f_color`, `f_boundary`, `f_hu`, `f_region`, and `f_texture` from each image, and stack those feature vectors in an $N \times 106$ feature matrix, where N is the total number of images, and 106 is the feature vector dimensionality. You will also create a corresponding $N \times 1$ label vector (actually a cell (list) as elaborated below). You will create two feature matrices associated with each image class, `F_train` and `F_test`, along with two label vectors, `y_train` and `y_test`. `F_train`, `y_train` will contain information about the first 90% of the images in a given directory and `F_test`, `y_test` will contain the remaining 10%. In assigning images to `F_train`, `y_train` or `F_test`, `y_test`, choose the next lowest integer (i.e., floor) of $N * 0.9$ for the train images and the remaining images for the test images. The label vectors `y_train` and `y_test` will be cells (lists) of the class strings (e.g., `'emu'`). Specifying the `'emu'` and `'flamingo'` directories, compute `F_train`, `y_train`, `F_test`, `y_test`. Print out the first column of `F_test` (as a row vector for space considerations). Print out `y_test` (as a row vector for space considerations).

(ii) Some of the features have a larger range than others. We don't want those features to have undue influence on the classification. We will thus normalize the feature matrices to have range [0,1]. There will be two slightly different procedures for normalizing `F_train` and `F_test`. You may write two functions `[F_norm,mx,mn]=normalize_Ftrain(F_train)` (`F_norm,mx,mn=normalize_Ftrain(F_train)`) and `F_norm=normalize_Ftest(F_test,mx,mn)`. For a (small) amount of extra credit, you may write one function `normalize_feature_columns` to cover both of the following procedures using the `varargin` (`argv`) capabilities of Matlab (python). To normalize `F_train`, from each column: subtract the minimum of the column and divide by the maximum of the column. Note that you will need to compute the maximum *after* you subtract the minimum in order to correctly normalize to [0,1]. Additionally, save the maximum values for each column in a 1×106 vector `mx` and the minimum values for each column in a 1×106 vector `mn`. To normalize `F_test`, from each column: subtract the corresponding minimum from `mn` and divide by the corresponding maximum from `mx`. This procedure treats the test data exactly the same as the training data. For the same `F_train`, `F_test` as in part (f-i), compute the normalized matrices `Fn_train`, `Fn_test`. Print every tenth entry in `mx` and `mn`, i.e., `mx(1:10:end)` and `mn(1:10:end)` (`mx[0::10]` and `mn[0::10]`). Print out the first column of `Fn_test` (as a row vector for space considerations).

(g) SVM classification.

(i) We will use a support vector machine (SVM) classifier here. You will declare and train your binary (two-class) classifier using the commands `clf=fitcsvm(Fn_train,y_train)` (`clf=svm.SVC(kernel='linear'); clf.fit(Fn_train,y_train)`). You will test your classifier by predicting the labels for the test data using `y_test_hat=clf.predict(Fn_test)`. Now that you have the predicted class labels `y_test_hat`, you can compare them to the known class labels in `y_test`. You can compute the confusion matrix for the classifier `C=confusionmat(y_test,y_test_hat)` (`sklearn.metrics.confusion_matrix(y_test,y_test_hat)`). You can compute the overall classification accuracy from the confusion matrix `acc=sum(diag(C))/sum(sum(C))` (`acc=np.diag(C).sum().astype(float)/C.sum()`). Using the emu/flamingo setup from part (e) above, print out the confusion matrix `C` and the accuracy `acc`.

(ii) You can train a multi-class SVM with the syntax `clf=fitcecoc(Fn_train,y_train)` (`clf=svm.SVC(kernel='linear'); clf.fit(Fn_train,y_train)`). The code for creation of the confusion matrix and classifier accuracy is the same. Define feature matrices for the object categories `'emu'`, `'flamingo'`, and `'strawberry'` and print out the confusion matrix `C` and accuracy `acc`.

(iii) Train an SVM to distinguish between an `'emu'` and a `'flamingo'` using only color features `f_color`, only boundary features `f_boundary`, only region features (`f_hu` and `f_region`), and only texture features `f_texture`. Print out the confusion matrix `C` and classification accuracy `acc` for each case. What do these metrics tell you about the relative discriminatory importance of each of those features? Does the classifier seem to err toward one particular class?

(iv) Train an SVM to distinguish between a `'chair'`, and a `'windsor_chair'` using all features, only color features `f_color`, only boundary features `f_boundary`, only region features (`f_hu` and `f_region`), and only texture features `f_texture`. Print out the confusion matrix `C` and classification accuracy `acc` for each case. What do these classification accuracies tell you about the relative discriminatory importance of each of those features? Are the important features here different than the emu/flamingo classification?

(v) Extra credit: Train an SVM to classify all 101 object categories and print the confusion matrix `C` and accuracy `acc`. There may be some fussy issues here like missing annotation files or empty annotations that will require some fussing with your code. Report the average classification accuracy `acc`.

Additionally, compute the confusion matrix `C`, normalize each row by the sum of the row (i.e., convert rows of `C` to percentage instead of count) and visualize the normalized `C` as an image. Which category/categories has/have the best classification accuracy? Which have the worst?

EE588 Advanced Image Processing
Project #4 Image Classification
(due Monday, Nov. 12, 2018, 10:30:00 am)

Confidence (number 0 to 100% about your confidence in your performance on this project and optional statement about areas where you are particularly confidence or not):

Difficulty (number 0 to 100%):

Time Spent (hours):

Problem	**Points**
(a) Dataset setup	/10
(b) Color features	/15
(c) Boundary features	/15
(d) Region features	/15
(e) Texture features	/15
(f) Set up feature matrix	/15
(g) SVM classification	/15
TOTAL	/100