Coursework Assignment

Deadline: 10th May 2019 at 4PM

This assignment brief was first released on 16th March 2019

The assignment grade, which is worth 50% of the total grade, is separated into 2 components: a **2-page report** (please use the provided **template**) and **source code**. The 2-page report should comprise a summary on your **image feature extraction steps** (e.g., colour histogram features, scale-invariant feature transform (SIFT) features, and/or convolutional neural network (CNN) features), and your implementation of **a zero-shot recognition system** based on a Direct Attribute Prediction (DAP) concept (**Lecture 13**).

DAP system models the probability of a certain object (e.g. polar bear) being present in the image using the probabilities of being present for each of the attributes that a bear is known to have. For example, if we detect the attributes "white", "furry", "bulbous", "not lean", "not brown" etc. in the image, i.e. attributes that a polar bear is known to have, we can be fairly confident that there is a bear in the image. Hence, we can recognize a polar bear **without** ever having seen a polar bear, if

(1) we know what attributes a polar bear has, and
(2) we have classifiers trained for these attributes, using images from other object classes (i.e. other animals).

Follow the steps below to implement a DAP-based zero-shot recognition system.

First, copy the Animals with Attributes dataset (originally appearing here[1]) from `http://users.sussex.ac.uk/~nq28/Subset_of_Animals_with_Attributes2.zip` (4.9GB). The dataset includes 50 animal categories/classes, and 85 attributes. The dataset provides a 50x85 `predicate-matrix-binary.txt` which you should read into **Matlab** using `M = load('predicate-matrix-binary.txt');` or into **Python** using `M = numpy.loadtxt('predicate-matrix-binary.txt')`. An entry `(i, j)=1` in the matrix says that the i-th class has the j-th attribute (e.g. a bear is white), and an entry of `(i, j)=0` says that the i-th class doesn't have the j-th attribute (e.g. a bear is not white).

Image set in JPEG format is provided. You should extract image features, the **options** are:

- **Colour** histogram features (image histogram **Lecture 4** and spatial pyramid of colour histogram **Lab Worksheet 6**). Think carefully about the appropriate colour space, e.g. RGB (Red Green Blue), HSV (Hue Saturation Value), Lab (Lightness green-red blue-yellow). For **Python**, remember that *OpenCV uses a BGR ordering of their image channels* (**Lab Worksheet 1**, Question 13).

- **SIFT** (Scale-Invariant Feature Transform) features/interest points (**Lecture 9**). You should use **Matlab**'s built-in functions from *the Computer Vision System Toolbox* to do this feature detection and extraction step

[1] https://cvml.ist.ac.at/AwA2/

(`https://uk.mathworks.com/help/vision/feature-detection-and-extraction.html`). For an example on how to extract SIFT/SURF feature detector and descriptor, type `openExample('vision/ExtractSURFFeaturesFromAnImageExample')` in the Matlab command window. SURF (Speeded-Up Robust Features) is simply a speed-up version of SIFT. In Matlab, the default feature size is 64, to make it 128, you can set the 'SURFSize' argument to 128 (`https://uk.mathworks.com/help/vision/ref/extractfeatures.html`). For **Python**, you should use the function `sift.detectAndCompute()` from the OpenCV Toolbox (for an example how to use the function please refer to `https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#sift-intro`). SURF is also available from the OpenCV, see the general description on feature detection and description at `https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html`. When using a SIFT/SURF feature extraction method, each image will have a variable number of SIFT/SURF feature descriptors, usually of size 128 each. You can aggregate all feature descriptors from multiple images, and perform K-Means clustering over all of them (**first explained on 15th March 2019, will be re-explained on 22nd March 2019**). An image can now be represented by the histogram over the K cluster centers, this is called **bag-of-visual-words** features.

- **CNN** (convolutional neural network) features (**Lecture 17 and 18**). You can extract image features using a pretrained convolutional neural network. In **Matlab**, you can use *the Deep Learning Toolbox*; for an example on how to use a pretrained AlexNet to extract features at 'fc7' layer, please refer to `https://uk.mathworks.com/help/deeplearning/examples/feature-extraction-using-alexnet.html`. In **Python**, you may refer to this Colab file `https://colab.research.google.com/drive/1GSCTjj32TrCRcsvREqUKZ3SY0LJeBDtm#scrollTo=FQ4Z9womVbfs&forceEdit=true&offline=true&sandboxMode=true`.

Your zero-shot recognition system should split the object classes (not images) into a training and test set. In this scenario, the training classes are animals that your system will see, i.e. ones whose images the system has access to. In contrast, the test set contains classes (animals) for which your system will never see example images. The 40 training classes are given in `trainclasses.txt` and the 10 test classes are given in `testclasses.txt`. (Use `[c1, c2] = textread('classes.txt', '%u %s');` in **Matlab**, or `c1 = np.loadtxt('classes.txt', delimiter='\t', usecols=[0])` and `c2 = np.loadtxt('classes.txt', delimiter='\t', usecols=[1], dtype=np.str)` in **Python** to read in the class names.) At each time, we will assume that a query image can only be classified as belonging to one of the 10 unseen classes, so chance performance (randomly guessing the label) will be 10%.

You will use **all** or **a random sample of all** images from the training classes (or rather, their feature descriptors) to train a classifier for each of the 85 attributes. The *predicate matrix* mentioned earlier tells you which animals have which attributes. So if a bear is brown, you should assign the "brown=1" tag to all of its images. Similarly, if a dalmatian is not brown, you should assign the tag "brown=0" to all of its images. You will use the images tagged with "brown=1" as the positive data in your classifier, and the images tagged with "brown=0" as the negative data, for the **"brown" classifier**. Use the **Matlab** `fitcsvm` function to train the classifiers. Save the model output by each attribute classifier as the j-th entry in a models cell array (initialized as models = cell(85, 1);). For **Python**, you can use `LinearSVC` function from the scikit-learn Toolbox (`https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`). Note that if you sample data in such a way that you have either no

positive or no negative data for some attribute classifier, you'll get a classifier that only knows about one class, which is a problem. However, for every attribute, there are some classes that do and some that don't have the attribute. So you just have to make sure you sample data from all classes, when training your attribute classifiers. You now have one classifier for each attribute.

You next want to apply each attribute classifier j to each image l belonging to any of the test classes. You want to save the *probability* that the j-th attribute is present in the l-th image. To do so, you have to do one extra operation to your classifier. In **Matlab**, for each of the j attribute classifiers, run the function `fitSVMPosterior` on them, i.e. call `model = modelsj; model = fitSVMPosterior(model); modelsj = model;` (or if you want, run this function on each classifier before saving it into the cell array). Then to get the probability that the l-th image contains the attribute j, call `[label, scores] = predict(model, x);` where x is the feature descriptor for your image. Then scores will be a 1x2 vector, check model.ClassNames to know which probability belongs to which class. Ensure that the probabilities sum to 1, by calling `assert(sum(scores) == 1)`, or if x contains the descriptors for multiple images, `assert(all(sum(scores, 2) == 1))`. In **Python**, for each of the j attribute classifiers, you can use `CalibratedClassifierCV` to get a probability output from `LinearSVC` function, i.e. call `svm = LinearSVC()`, then `clf = CalibratedClassifierCV(svm)`, and finally `clf.fit(X_train, y_train)` with the appropriate collection of training inputs X_train and training outputs y_train. Then to get the probability that the l-th image contains the attribute j, call `y_proba = clf.predict_proba(x)` where x is the feature descriptor for your image. Save these probabilities so you can easily access them in the next step.

You will now actually predict which animals are present in each test image. To perform classification of a query test image, you will assign it to the test class (out of 10) whose attribute "signature" it matches the best. How can we compute the probability that an image belongs to some animal category? Let's use a toy example where we only have *2 animal classes* and *5 attributes*. We know (from a predicate matrix like the one discussed above) that the first class has the first, second, and fifth attributes, but does not have the third and fourth. Then the probability that the query image (with descriptor x) belongs to this class is $P(class = 1|x) = P(\text{attribute\_1} = 1|x) \times P(\text{attribute\_2} = 1|x) \times P(\text{attribute\_3} = 0|x) \times P(\text{attribute\_4} = 0|x) \times P(\text{attribute\_5} = 1|x)$. The "|x" notation means "given x", i.e. we compute some probability using the image descriptor x. Let's say the second class is known to have attributes 3 and 5, and no others. Then the probability that the query image belongs to this class is $P(class = 2|x) = P(\text{attribute\_1} = 0|x) \times P(\text{attribute\_2} = 0|x) \times P(\text{attribute\_3} = 1|x) \times P(\text{attribute\_4} = 0|x) \times P(\text{attribute\_5} = 1|x)$. You will assign the image with descriptor x to that class i which gives the maximal $P(class = i|x)$. For example, if $P(class = 1|x) = 0.80$ and $P(class = 2|x) = 0.20$, then you will assign x to class 1. How do you compute $P(\text{attribute\_}i = 1|x)$? This is a probability value you've computed already.

You will classify **each test image from the 10 unseen (test) classes**, and compute the average accuracy. What to include in your submission:

1. A function `extract_feature(...)` that outputs a Dx1 array (where D is the dimensionality of your features) for each image. For example, for a colour histogram feature with 8 bins in Hue, 4 bins in Saturation and 4 bins in Value, you will have $D = 128$. For a bag-of-visual-word feature using SIFT/SURF detectors/descriptors, you will have $D = K$ where K is the number of clusters in the K-Means clustering algorithm. For a CNN feature with a pretrained AlexNet at the 'fc7' layer, you will have $D = 4096$.

2. A function `train_attribute_models(...)` that outputs 85 attribute classifier models.

You are free to pass in whatever arguments you need, and are welcome to add any additional outputs after models.

3. A function `compute_attribute_probs(...)` that outputs a $85 \times Ntest$ matrix of probabilities, where $Ntest$ is the number of test images; with the (j, l) entry of the matrix as the probability that the j-th attribute is present in the l-th image. Again, use any inputs you like, and any additional outputs after the first one.

4. A function `compute_class_probs(...)` that outputs an $10 \times Ntest$ matrix of probabilities, with (i, l) entry of the matrix as the probability that the i-th class is present in the l-th image.

5. A function `compute_accuracy(...)` that outputs a single real number denoting the overall accuracy of your system, averaged over the Ntest test images. **Also include the overall accuracy score in your 2-page report**.

2-page report guide

25 points **Outline of methods employed**
This does not have to be in depth, and I do not expect you to regurgitate the contents of the lecture notes. You should state clearly what methods you have used, what parameters you have used with those methods and what the purpose of these methods were. If you have developed any of your own approaches, or you have adapted either a built in approach, or improved on it, then you should discuss that here.

40 points **Results achieved and analysis**
In this section, you should present your results by stating the accuracy results for the 10 unseen (test) classes. Here you should consider which techniques or approaches worked well, and which did not. For example, you can **compare** the performance of your zero-shot learning system based on **different features** (colour histogram, SIFT, or CNN features). You can also **combine** multiple features. Remember that this is a difficult problem, and it is unlikely that you results will be perfect. It's OK to have a poor result, as long as you can explain why you think this happened. Your duty is to report the truth as it happened in an objective manner.

35 points **Discussion**
You should also take the opportunity to discuss any ways you can think of to improve the work you have done. If you think that there are ways of getting better performance, then explain how. If you feel that you could have done a better job of evaluation, then explain how. What lessons, if any have been learnt? Were your goals achieved? Is there anything you now think you should have done differently?

Marking

General marking criteria for Computer Vision is available below. Marks will be roughly equally divided between the code and the report (but only roughly, because good comments in the code can mean less need for detail in the report). Programs that perform accurate zero-shot learning are, of course, likely to do well. However, the quality of the code and report will be important factors regardless of performance, and originality and ingenuity will be taken into account.

70% – 100% **Excellent**

Accurate predictions (on the 10 unseen test classes) underpinned by an excellent workflow, code appropriately commented, evidence of critical assessment of the model. The work shows very good understanding supported by evidence that you have extrapolated from what was taught, through extra study or creative thought. Work at the top end of this range is of exceptional quality.

60% – 69% **Good**

Reasonable predictions (on the 10 unseen test classes) underpinned by a solid workflow, code appropriately commented, some attempt to critically assess the model. The work will be very competent in all respects. Work will evidence substantially correct and complete knowledge, typically not going beyond what was taught.

50% – 59% **Satisfactory**

Reasonable predictions (on the 10 unseen test classes) underpinned by a reasonable workflow, little or no critical assessment. The work will be competent in most respects but there may be minor conceptual errors or methodological oversights.

40% – 49% **Borderline**

Average predictions (on the 10 unseen test classes) underpinned by the most basic workflow, no critical assessment of the work. The work will show the most basic understanding of fundamental concepts acceptable at this level.

30% – 39% **Fail**

Poor predictions underpinned by inadequate workflow, flawed implementation, no evidence of critical assessment of the model. The work will show inadequate knowledge of the subject, is seriously flawed and displaying major lack of understanding.

Below 30% **unacceptable (or not submitted)**

Work is either not submitted or, if submitted, very seriously flawed.

References

[1] Yongqin Xian, Christoph H. Lampert, Bernt Schiele, Zeynep Akata. Zero-Shot Learning - A Comprehensive Evaluation of the Good, the Bad and the Ugly. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2018. `https://arxiv.org/abs/1707.00600`.