TTIC 31040, Spring 2021
Project: texture classification

May 22, 2021

1 Project goals

We will construct a texture classifier that will take an image from one of 47 types (categories) and assign it to the most likely type. This is a multi-class image classification task, mapping an image to a single label.

We have provided the data from the Describable Textures Dataset (DTD). You can learn more about it at

https://www.robots.ox.ac.uk/ vgg/data/dtd

however you should not need to donwload anything besides what we have included in the `.zip` file for this project. Once expanded, the folder `dtdata` will contain a file tree, under `dtdata/images`, where each su-folder contains examples of texture of a given class. You will also fine in `dtdata/train.txt`, `val.txt` and `test.txt` the lists of train/val/test portions of the data set (splitting the data into non-overlapping portions, each one third of the total). Please respect this split: use training to extract any data and to train the classifier, validation to debug your methods and to evaluate/tune hyperparameters, and test to report final results (ideally only touching it once).

2 Steps to implement

1. Implement basic data ingestion code, allowing you to load the images, organize them, and map them to numerical or one-hot representations of the categories. There is no particular mapping you would need to stick with, so you can make it alphabetical or anything else; but make sure you can also map back from category labels to texture category names, for display and analysis purpose.

   Note that images in DTD do not have a fixed size. This should not be an issue for you, since the method we are going to focus on below does not require treating a set of images as a single tensor. You *may* need to represent a set of images as a matrix of feature vectors, but those feature vectors, i.e., visual word assignment histograms, will be of a fixed size regardless of the size of the input images in pixels.

2. Construct a filter bank. You can use an existing implementation of a common filter bank, such as this one for the Leung/Malik filter bank:
   https://github.com/tonyjo/LM_filter_bank_python

   of design and build your own. At the end you should have code that takes an image and produces for every pixel (i, j) (or every pixel at a given stride, e.g., every 2nd pixel or every 3rd pixel in both dimensions) a vector in $\mathbf{x}_{i,j} \in \mathbb{R}^f$ where f is the number of filters in the bank, and $x_{i,j,s}$ is the value that the filter s in the bank produces when centered at (i, j).

3. Implement k-means clustering of the response vectors. You can write your own or use a standard implementation.

   A tip: k-means could take a while to converge on a really large data set. You can speed things up by the following (heuristic) trick: Start by running k-means on just 1% of the data, randomly selected. This will be fast, but of course the results may be noisy. Then run k-means on a bit more data (say, 3%) initalizing the algorithm with the output (the means) of the 1% run. Continue running it on increasing fraction of the data (5%, 10%, 25%), each time initializing from the output of the previous run. Typically, subsequent runs will require fewer and fewer iterations to converge, since your initial estimate of the means will be already increasingly close to the optimal. The final run on the full dataset often takes just one iteration, after which the assignments remain fixed and thus the algorithm terminates.

4. Implement mapping from an image to texton assignment histogram, using the machinery you build above. Given an image and a dictionary (constructed by k-means), this would first compute the set of filter responses (you can make the stride at which you compute this an input argument of the function), then assign each of them to the "word" in the dictionary, and finally compute a histogram of assignments.

5. The next step is to construct a classifier, that given a texton assignment histogram assigns the image to one of 47 texture classes.

   You can choose the classification method as you see fit; keep in mind that the number of examples is relatively small, so you probably can't use a complex classifier (like a neural network). Some obvious choices include K nearest neighbor classifier (for some small K, using an appropriate distance measure, such as χ^2) and a linear (softmax) classifier directly from the histograms; think about the need to regularize it, taking into account the dimensions of feature vectors fed to the classifier and the amount of training data.

6. Finally, write the code for training the classifier and testing it on validation data and, eventually, on the test data, yielding a confusion matrix and accuracy. You should report the overall accuracy and the most commonly confused categories.

3 Project submission

Please describe the pipeline you end up implementing in the PDF writeup. We recommend LaTeX, but other typesetting environments like Word or Google Docs are fine, as long as the submitted document is in PDF. Please submit this PDF (try to keep it shorter than 4 pages) along with the full notebook, and a `.zip` file containing any additional material we'd need to replicate your experiments. This may include your own images; Python code written in separate files imported into the notebooks (if you can avoid that, please do, as it makes our work easier); etc.

Your writeup should include:

- Precise description of the implementation, written in a way that would allow a fellow student in the course to replicate your design.
- Discussion of any choices you had to make, and an explanation of how and why you made a particular choice there. (E.g., things that can be parameterized in different way; settings for hyperparameters; etc.)
- Qualitative evaluation of the performance of your method, and any thoughts on potential improvements one could make with more time/effort. This includes improvements to the method per se, as well as improvements to implementation (such as speeding it up).
- Discussion of the fundamental limitations of the method as designed and implemented, including any specific failure modes (types of input/reference combinations in which the method is likely to fail). In particular, if you notice some pairs (or larger sets) of texture categories in DTD that are commonly confused, can you explain this? and perhaps propose a remedy (even if you do not implement it)?

For two-person projects: Please include analysis of the performance of your method as a function of the number of training samples per class (going from 1 example per class to the N you decided on when splitting the data). Also, please experiment with parameters of the filter bank and k used in k-means, with at least two meaningfully different regimes for both (e.g., doubling/halving the number of filters; increasing k by a factor of at least 2; adding different types of filters to the bank, etc.) and report the results and conclusions. Finally, experiment with your classifier on images that are not from the dataset. Find your own examples that you think match some categories in the dataset; try it on images not of textures, but where you perhaps feel you can predict the result.

For one-person projects: You can pick a single setting for all the hyperparameters (please explain how you picked it), and do not need to report and discuss results on anything other than the test portion od DTD.