Introduction

This is a brief report on the methods used in the TagMe competition. More about the competition at http://events.csa.iisc.ernet.in/opendays2014/events/MLEvent.

Software Used

The feature extraction was done in MATLAB, while all the data manipulation and classification was done in Python. All computation was done on a Macbook Air (1.8 Ghz, 8 GB RAM).

Feature Extraction Process

Instead of spending time coding up expensive feature computations, we decided to do unsupervised feature selection. Because the test set will be released only a day before submission, it was decided that the whole process should not take a lot of time to give enough room for testing, validation and interpretation.

However, instead of using the image pixel values directly, we decided to do unsupervised feature learning on one of the best off-the-shelf feature for object recognition - Scale Invariant Feature Transform (**SIFT**). We decided to do dense SIFT using the VLFeat Library, which has a well tested MATLAB interface. This is distributed under a BSD license. Dense SIFT was computed with a patch size of 8 and a between patch distance of 4. No image resizing was done. Images were smoothed before computing SIFT features. The VLFeat implementation is very scalable.

For actually learning features, we used a modern method, quite popular in Deep Learning today. Instead of resorting to sparseRBMs or spare Coding methods which require careful tuning of a lot of hyper parameters, we used **sparse Filtering**, a technique introduced by Jiquan Ngiam et. al[1], which only has two hyper parameters to tune, namely the number of layers and the number of units in each layer. The MATLAB code is openly available and uses the minFunc toolbox for optimization. Simply put, instead of trying to attempting to construct the true data distribution, spare Filtering minimizes a single cost function, the sparsity of $L2$ normalized features. Ease of use and scalability were the reasons for this choice.

We experimented with both single layer and two layer networks, with a feed forward step. Details are in the paper [1].

The first layer was learned with 128 bases to provide for an almost over-complete representation, the apparent benefits of which are well discussed in Deep Learning/Compressed Sensing communities. Since the SIFT features are $128XN$, we used 128 units in the first layer of the network and 64 units in the second layer.

Since single layers seem to have better performance [2], we used a single layer with 128 bases instead of multiple layers. A simple experiment with two layers (64 bases in second) lead to slightly lower performance on the validation set.

Similarity/Distance Measures

Since we don't have any kernels, we did not use any type of similarity metrics.

Classifier

We passed the learned features through a **multi-class SVM with a linear kernel**. The intuition was that if the learned dictionary was good enough, then the data points should be linearly separable.

References

- [1] *Sparse Filtering*, Jiquan Ngiam, Pang Wei Koh, Zhenghao Chen, Sonia Bhaskar, Andrew Y. Ng; **NIPS 2011**.
- [2] *An analysis of single-layer networks in unsupervised feature learning*, A Coates, AY Ng, H Lee; **ICML 2011**.

Feature Extraction/Algorithm Details

All details of the algorithm and feature selection are already given above. The code is open source at https://github.com/john316/tagme. The code was open sourced only after the contest ended, on 15th March 2014 IST.

One detail that we should mention is that the weights on SIFT features were pooled using a simple max pooling method which is empirically known to work better than average pooling. This substantially decreased the number of features, and actually improved prediction accuracy. We tried pooling sizes of 4, 8 and 16, and since 8 seemed to give the highest accuracy on the validation, we chose 8.

Validation and Parameter Tuning

The Validation set looked noisy with scaled, rotated and blurry images, so it was not surprising that unsupervised-learned dictionary bases could only reach around 80% to 82% on the validation set. The validation set was not used in the dictionary learning process to prevent over fitting. It was surprising that the leader board was rich with so many prediction accuracies reaching the high 90s and even 100!.

Prediction Algorithm

We tried Random Forests also initially, but they did not seem to perform as well as multi-class SVMs. The multi-class SVM we used is the sklearn-SVC implemetation from scikit-learn which is again open-source.

We checked if the provided features (which are eigenvalues of the image matrix in grayscale) combined with the learned features did a better job, and in fact it did. So the final feature set passed to the classifier was a simple combination of both types of features.

The final classification accuracy on the validation set was 85.2%.

Discussion

Why do you think your algorithm got the accuracy that it did on the validation data? Are there scope for improvements?

- Sparse Coding is known to work well with local feature descriptors, and hence it was no surprise that we were getting decent classification accuracies.
- With SIFT features instead of raw image pixels, it is known that better predictions can be made.
- The final choice of using all images from the validation set was a little arbitrary, since we had no way of verifying if it was over fitting or not; especially since the validation data was very noisy.
- There is a definitely a lot of scope for improvement, and most of these improvements were not tried in the interest of not wrecking my laptop!
- We did not experiment with bigger code book size for the final model, because of memory and time constraints. Bigger code books are know to learn much better representations in general, up to a certain size.
- Instead of using a naive max pooling technique, we believe a spatial pyramid learned on top of learned SIFT feature weights would have given slight boosts. In the interest of time, we did not do this.
- Another idea is to use a neural net available in Vowpal Wabbit since it might learn a better linear decision surface than the SVM. Since VW is built only for binary classification, we'd have to adapt it to perform multi-class classification. In the interest of time, we could not do it.