Data challenge - Kernel methods (MVA)

Team name : "Les dénoyauteurs"

Friday 8th April, 2022

Gonzalo REINOSO
École Normale Supérieure Paris-Saclay
greinoso@ucm.es

David SOTO
École Normale Supérieure Paris-Saclay
david.soto.c17@gmail.com

Abstract

This written report presents our approaches and the methodology we used for the data challenge as well as the results obtained. In this challenge, we experiment with various kernel methods to perform image feature extraction and classification.

1. Introduction

The data challenge consists of a simple image classification task. There are 10 different classes of images and each image/data is represented by a $32 \times 32 \times 3 = 3072$ dimension vector (where 32×32 is the size of the image and 3 is the number of channels). The train set contains 5000 classified images and the test set contains 2000 images to be classified. The goal of the data challenge is to implement machine learning algorithms without using classical machine learning libraries such as scikit-learn or libsvm, and to use these home-made algorithms to perform image classification and obtain the highest possible accuracy on the test set. Our team's name on the leaderboard is *"Les dénoyauteurs"*.

2. Methodology

First, we visualized the images to make sure we were properly reading the data and then, we tried different methodologies to perform feature extraction and classification. For feature extraction, our efforts focused on Scale-Invariant Feature Transform (**SIFT**) [3], Kernel Principal Component Analysis (**KPCA**) and Histogram of Oriented Gradients (**HOG**) [2]. We performed a preliminary experiment using built-in functions in order to check which methods performed well, this pointed us towards experimenting with SIFT and KPCA. Before passing these features to our **SVM**, we noticed that it is **crucial** to **scale and center** them, at first, we skipped this step and we were not managing to get more than 25% of images correctly classified.

2.1. Sanity check and visualization

We start by checking that the training data are uniformely distributed on classes. It is the case. Each class (aircraft, car, bird, cat, deer, dog, frog, horse, boat and truck) in the training set contains 500 samples. Afterwards, we display some images of the train set and the test set to have an idea about their appearances. As we can see in Figure 1, the images are challenging. The images appear blurred and it is sometimes difficult to "understand" the image.

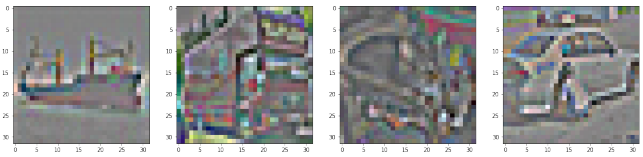

Figure 1. Visualization of some images of the train set

2.2. KPCA vs HOG vs SIFT features

First of all, we tried to reduce the dimension of the data by using KPCA, HOG and SIFT. As mentioned, we first experimented with built-in functions in order to get an idea of these methods' performance and to know in which one should we focus our efforts. We observed that HOG and KPCA gave slightly worse results than SIFT when choosing the best parameters through a grid search (different grid search for each method since the parameters to be tuned are different). Thus, we implemented a SIFT algorithm from scratch adapting it from the matlab implementation of Svetlana Lazebnik (University of Illinois) as well as a KPCA. Then, we checked our implementation's performance choosing the best parameters through intuition (motivated by the small image size) and a grid search. As we expected, KPCA performed worse than SIFT.

Since the data comes from images, we found it logical to focus on a feature extractor specific for this type of data: SIFT. One of the key steps of the method is the construction

of the SIFT descriptors. The idea is that if two images are similar, they will have similar SIFT descriptors, otherwise the descriptors will be different. The first step of a regular SIFT algorithm consists in finding the key points of an image, which mainly correspond to the objects' edges and corners in an image. In our code, we did not implement this part but instead we simply created a grid on the images where each element of the grid serves as a reference point for a SIFT descriptor. We tuned SIFT parameters such as contrast threshold intuitively at first and then adjusted it empirically based on performance on a validation set. Finally, extracting SIFT features from the images enables us to have less but more relevant features to be fed to the classification algorithms.

2.3. Kernels

In our approach and implementation, we used the following classical kernels :

- **Linear kernel** : $K(x, y) = x^T y$
- **RBF kernel** : $K(x, y) = exp(-\gamma \left\| x - y \right\|^2)$
- **Polynomial kernel** : $K(x, y) = (x^T y + p)^d$ where d is the degree and p a constant (in our code the parameter `coef0` represents the constant p)

The implementation of these kernels can be found in the folder containing our scripts (code). Our experiments showed that the RBF kernel had the best performances.

2.4. Classification algorithms with kernels

The kernels presented above have been used in the SVM classifier, which we have implemented (see the code). We implement a "one vs all" SVM approach instead of the usual "one vs one" approach because the latter takes too much time without necessarily performing well. Indeed, for a "one vs all" approach, we train our model using K = 10 classifiers whereas the "one vs one" approach would have required K(K-1)/2 classifiers, that is 4.5 times more than the "one vs all" approach since we have K = 10. In order to solve the convex optimization, we use QP solver (from the `cvxopt` package) which stores the entire kernel matrix. Indeed, our implementation of a multi-class SVC classifier uses a QP solver that comes from the `cvxopt` library.

2.5. Data augmentation

We performed data augmentation on the training set. This consists in performing some transformations on the existing images and to add the transformed images to the training set. This technique would help the classification algorithm to better generalize. A classic method for data augmentation consists in carrying random rotations of the images and changes in scale. However, SIFT features are already invariant to rotations and changes in scale. Augmenting the training set with rotations and changes of scale creates duplicates of the SIFT features and thus would not help. Hence, we chose to separately use gaussian blur and horizontal flip for data augmentation, each of these methods allowed to double the size of the training set. We did not augment the dataset with both methods at once because 15000 images would have been intractable (with our resources) for our SVM. The extension of the train set enhanced the performance of the classification algorithm. Using data augmentation, we obtain a score of 63.1% on the public leaderboard, which is an improvement from our previous score of 58.8% obtained without data augmentation. Both augmentation methods improved the score, had the execution time been shorter, we would have liked to play more with the parameters to find the optimal combination.

3. Final model

We chose the method that gave us the best results on the training data. Our final model uses a RBF kernel for SVM (using a QP solver), applied to SIFT features. The parameters of the final model are : $\gamma = 0.001$ (parameter of the RBF kernel) and $C = 7.5$ which is the margin regularizer. The choice of C was key to prevent over/underfitting. Our final model uses data augmentation with a horizontal flip of images. Therefore, the training set size is doubled. After having set the SIFT parameters, we extract the SIFT features before using a multi-class SVM (one-versus-all) with a RBF kernel to classify the images. Extracting the SIFT features takes about 30 minutes when using data augmentation, otherwise it takes 15 minutes. Afterwards, we train our classification model on the training set using our function `fit` that we have implemented. This step includes a projection of the training data in order to check how much of the training data is correctly classified. A very high/low accuracy could indicate overfitting/underfitting. This step takes about 3 hours when using data augmentation, otherwise it takes around 25 minutes.

4. Results and conclusion

Us, a.k.a. *"Les dénoyauteurs"*, obtained a score of 0.631 on the public leaderboard, which ranked us 7th out of 53 participants. We obtained a final score on the private leaderboard of 0.619 which ranked us as well 7th out of 53 participants. Other interesting approaches could improve performance. First, different kernels [1] adapted to this scenario could have been used and a voting method among different predictors seems as well like a reasonable approach.

Our code is available here: code. The notebook reproducing our best submission is `Main.ipynb`.

References

[1] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Kernel descriptors for visual recognition. In *NIPS*, 2010. 2

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005. 1

[3] G LoweDavid. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004. 1