# Kaggle Challenge Report: Image Classification with Kernel Methods

Team barthoss
Barthélémy Dang-Nhu
ENS Paris-Saclay
barthelemy.dang-nhu@ens-paris-saclay.fr

## Abstract

This document is a report of the Kaggle challenge of the course "Kernel methods"[1]. The challenge was an image classification problem with 5 000 images for the train set perfectly balanced among ten classes and 2 000 images for the test. Each images are RGB images of size $32 \times 32$. The key ingredient of this challenge was to only use machine learning methods based on kernel. The code is available at https://github.com/barthoss/kernelmethod.

## 1 Kernels

For this project we use the three following kernels functions:

- **Polynomial kernel**:

$$K(x, y) = \exp\left(-\gamma \|x - y\|_2^2\right)$$

  which has one hyper-parameter.
- **RBF kernel**:

$$K(x, y) = (\gamma \langle x, y \rangle + r)^d$$

  which has three parameters.
- **TanH kernel**:

$$K(x, y) = \tanh(\gamma \langle x, y \rangle + r)$$

  which has two parameters

## 2 Features Extractors

We want features extractor that transform images to vector. Three of the following ideas were implemented.

### 2.1 Raw data

The first simple idea was to use raw data, each image RGB channel of the image is flatten and concatenated together. The final vector for each image is of size $32 \times 32 \times 3 = 3072$.

### 2.2 Pooling

Here we use pooling on square cells of each image. There are three hyper-parameters:

- The aggregator function: `mean`, `min` or `max`.
- Size: the size of the square cell to aggregate.
- Stride: the shift between each square cell.

The pooling is performed independently on each RGB channels and the pooled images are the flatten and concatenated in one vector. The final size of the feature vectore is of size $3\left(\left\lfloor \frac{32 - size}{stride} \right\rfloor + 1\right)^2$.

### 2.3 Histogram of Oriented Gradients (HOG)

We implemented Histogram of Oriented Gradients[3], but first we needed to transform colored images to black & white images. To do so we performed a PCA on all the pixels of the training set as RGB three sized vectors. It appears that 81% of the variance is along the principal component of eigen vector $(0.582, 0.583, 0.566)^T$. Projecting images on this subspace is equivalent to average the RGB values, which is what we did.

Then we implemented HOG which calculates gradients of pixel intensities in an image, bins them by orientation, computes histograms within cells, and concatenates them into blocks for normalization. This generates a compact feature descriptor capturing local edge information, commonly used in object detection tasks in computer vision. The parameters are the size of the blocks and the number of orientation bins, here the stride is equal to the size. Therefore the size of the final vector is $\left(\left\lfloor \frac{32 - size}{size} \right\rfloor + 1\right)^2$.

## 3 SVMs

For this project we use SVM for classification. SVM are originally used for binary classification, that why we user four differents methods for multi-class classification:

### 3.1 1 VS. Rest

We train a SVM $f_c$ for each class $c$ where the labels are 1 for any item of class $c$ and $-1$ for other items. We then use the following prediction function:

$$\hat{y}(x) = \text{argmax}_c \; f_c(x)$$

There are in total $C$ SVM trained on the $N$ items.

### 3.2 1 VS. 1

Here we train one SVM $f_{c/c'}$ for each pair of class $c, c'$ where labels are 1 for item of class $c$ and $-1$ for items of class $c'$, other items are discarded. We then use two different methods for prediction. One called *sum*:

$$\hat{y}(x) = \text{argmax}_c \sum_{c' \neq c} f_{c/c'}(x)$$

| | Polynomial | | | RBF | | | TanH | | |
|---|---|---|---|---|---|---|---|---|---|
| | None | Pooling | HOG | None | Pooling | HOG | None | Pooling | HOG |
| SVM 1v1 vote | 0.329 | 0.364 | 0.473 | 0.119 | 0.236 | 0.397 | 0.173 | 0.201 | 0.273 |
| SVM 1v1 sum | 0.322 | 0.338 | 0.471 | 0.119 | 0.236 | 0.397 | 0.173 | 0.201 | 0.273 |
| SVM 1vR | 0.232 | 0.350 | 0.469 | 0.134 | 0.240 | 0.366 | | | |
| SVM CS | 0.272 | 0.354 | **0.478** | 0.206 | 0.353 | 0.476 | | | |

The other called *vote*:

$$\hat{y}(x) = \text{argmax}_c \sum_{c' \neq c} \mathbb{1}(f_{c/c'}(x) > 0)$$

Because $f_{c/c'} = -f_{c'/c}$ there are $C(C-1)/2$ SVMs trained on $2N/C$ items.

### 3.3 Crammer and Singer

In the previous methods, each SVM $f_c$ is learned independently. As proposed by Crammer and Singer[2] we implement SVMs jointly where we want $\forall c' \neq y_i,\ f_{y_i}(x_i) > f_{c'}(x_i)$. Therefore we have the following problem with $\phi$ the hinge loss.

$$\min_{f_1,\cdots,f_C \in \mathcal{H}} \frac{1}{n} \sum_i \phi\left(f_{y_i}(x_i) - \max_{c' \neq y_i} f_{c'}(x_i)\right) + \frac{\lambda}{2} \sum_c \|f_c\|^2_{\mathcal{H}}$$

It is shown that this problem is equivalent to

$$\min_{\alpha_1,\cdots,\alpha_C \in \mathbb{R}^n} \frac{1}{2} \sum_{i,j} \sum_{c,c'} K(x_i, x_j) \alpha_{c,i} \alpha_{c',j} - \sum_i \alpha_{y_i,i}$$
$$\text{s.t.} \begin{cases} \forall i,\ \alpha_{y_i,i} \leq 1/n\lambda \\ \forall i\ \forall c' \neq y_i,\ \alpha_{c',i} \leq 0 \\ \forall i,\ \sum_c \alpha_{c,i} = 0 \end{cases}$$

It requires to solve a quadratic equation with $nC$ variables which is unpractical. Therefore they proposed an alternative algorithm that iteratively solves problems of $C$ variables. With $f_c : x \mapsto \sum_i \alpha_{c,i} K(x, x_i)$ we can use the same prediction function as in 3.1.

## 4 Parameters Tuning

For each combination kernel/features extractor there are between two and six hyper-parameters. A first pre-tuning is done on the distance to set prediction to quickly have an order of magnitude a good hyper-parameters. The prediction function is the following:

$$\hat{y}(x) = \text{argmin}_c\ d_K(x, \mathcal{S}_c)$$

with

$$d_K(x, \mathcal{S}) = \sqrt{K(x,x) - \frac{2}{n} \sum_{y \in \mathcal{S}} K(x,y) + \frac{1}{n^2} \sum_{y,y' \in \mathcal{S}} K(y,y')}$$

Then we use fine tuning on hyper-parameters for each methods. Fitting an SVM can take a long time, so a timeout has been implemented to increase the number of combinations tested.

## 5 Results

We present the validation accuracy on the above table. It should be noted that each method takes a different amount of computation time, so there were not as many tests during fine-tuning. Therefore, the table may not necessarily represent the full potential of each method but rather a representation of what is quickly achievable.

One last method was tried, we used 1v1 vote SVM were for each pair of class we take the kernel/extractor having the best 1v1 validation accuracy, it was mostly a polynomial kernel with an HOG extractor. This method performs a validation accuracy of **0.527** however the validation set was used for the selection of the models and the evaluation, therefore the score is overealuated.

## 6 Conclusion

Finally we performed a Kagggle score of **0.475** thanks to the Crammer & Singer SVM and **0.469** thanks to the multi kernel SVM 1v1 with vote.

Some extensions can be tried:

- Use of a weighted SVM for the unbalanced problem 1vR. However it would only move each separating hyper-plane toward the examined class and we think it would not change the final prediction.
- Use different kernel methods such as Kernel Ridge Regression or Logistic Regression.
- At each submission the kaggle score was below the validation score, one idea would be to use cross-validation to have a more representative score during training.
- Use of different extractor, we tested a SIFT extractor however as the images were really smalls, the extractor struggles to extract more than 3 key-points even with exaggerated thresholds.

## References

[1] Michael Arbel. 2024. Data Challenge - Kernel methods (2023-2024). https://kaggle.com/competitions/data-challenge-kernel-methods-2023-2024

[2] Koby Crammer, Yoram Singer, Nello Cristianini, John Shawe-Taylor, and Bob Williamson. 2002. On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *J. Mach. Learn. Res* 2 (01 2002).

[3] N. Dalal and B. Triggs. 2005. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Vol. 1. 886–893 vol. 1. https://doi.org/10.1109/CVPR.2005.177