Classification d'image par méthodes à noyaux

Challenge Kaggle

Rapport de projet – Kernel Methods

Evann Courdier	Joseph Lam	Mathilde Bateson
ENS Cachan	ENSAE	ENS Cachan
evann.courdier	joseph.lam	mbateson
@ens-cachan.fr	@ens-cachan.fr	@ens-cachan.fr

Résumé

Ce rapport de projet présente notre méthodologie ainsi que les résultats obtenus pour le challenge Kaggle du cours de méthodes à noyaux du Master MVA, qui portait sur la classification d'images.

1 Présentation du problème

Le problème consistait à classer 10 types d'images, en utilisant entre autres des méthodes à noyaux. L'ensemble d'entraînement est composé de 5000 images, et l'ensemble de test de 2000 images. Chaque image de 32x32 pixels était représentée par un vecteur de longueur 3072 représentant les intensités des pixels sur les trois canaux de couleur rouge, bleu, vert.

Exploration des données Un affichage des images a révélé qu'elles avaient été pré-traitées, de sorte qu'elles étaient difficilement reconnaissables. Cependant, en les affichant en nuances de gris, on reconnaît les contours d'objets connus (voitures, avions, animaux...).

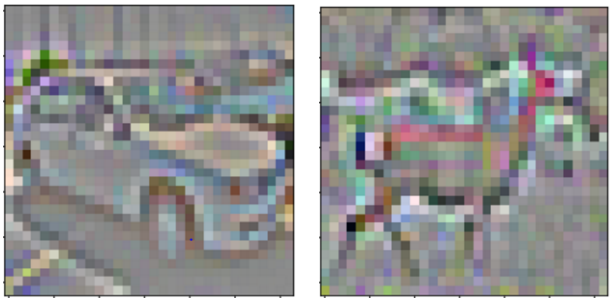

FIGURE 1 – Exemples d'images d'entraînement

2 Les différentes méthodes envisagées

Quels "features" extraire des images ? Une des premières idées qu'il vient est d'utiliser les statistiques de base de l'image de type histogramme des couleurs. Cependant, les résultats obtenus avec ces méthodes seules sont mauvais.

Nous avons donc appliqué des méthodes classiques d'analyse d'image, en commençant par utiliser le gradient de l'image. Ceci revient en réalité à effectuer la convolution de l'image par un filtre (ou noyau) de type $[-1, 1]$ sur x et $[-1, 1]^T$ sur y.

Nous avons ensuite testé l'application de différents filtres classiques (Sobel, Scharr, Laplacien, ...) et aussi de plusieurs filtres personnalisés (détecteurs de lignes, de croix..) de différentes tailles. Les filtres seuls ne donnaient pas de bons résultats, mais en en combinant un certain nombre (c'est à dire en convolant les filtres avec l'image et en concaténant les sorties), nous avons réussi à obtenir des premiers résultats intéressants autour de 45% de précision.

Cependant, ces résultats ne permettent pas d'être invariant aux petites translations de l'image. Nous avons alors testé un modèle de représentation de l'image prenant en compte ces translations : le HOG (pour Histogram Of Gradient). Nous avons tenté d'utiliser le HOG sur les différents canaux de l'image couleur ainsi que sur l'image en nuance de gris. Dans le HOG, c'est le gradient qui est utilisé, mais nous avons également tenté d'utiliser un autre noyau sur l'image, et il s'est avéré que l'histogramme d'orientation avec un filtre de Scharr donnait les meilleurs résultats.

Nous avons aussi essayé d'utiliser le principe de regroupement par blocs du HOG mais en utilisant uniquement l'histogramme d'intensité des réponses aux filtres, ce qui permettait de conserver l'invariance aux petites translations pour des filtres uniques. Cependant, bien que ces features soient relativement pertinents, leur utilisation en combinaison d'un noyau dans l'étape de classification donnait de mauvais résultats.

Nous avons également envisagé d'utiliser la méthode SIFT pour essayer de faire correspondre les images d'une classe entre elles mais cela requérait un important effort d'implémentation pour des résultats qui ne promettaient pas d'être meilleurs et nous avons choisi de concentrer nos efforts ailleurs.

L'aspect multi-échelle des features extraits est un aspect important et une possibilité pour prendre cela en compte est de concaténer des features calculés à différentes échelles. Nous avons fait de nombreux essais et la combinaisons la plus pertinente semble être la concaténation de deux hogs à deux échelles légèrement différentes (4 cellules de 8 pixels

et 3 cellules de 10).

Enfin, il convient de mentionner que nous avons essayé des techniques de réduction de dimension de type ACP et ACI, sans amélioration des résultats. Cela peut être du à la non linéarité de l'espace dans lequel se trouve les données, une variété de dimension élevée et non linéaire, ou l'information est répartie dans de nombreuses dimensions.

Quel noyau utiliser ? La classification immédiate des features extraits ne donnait pas de bons résultats. Nous avons donc considéré l'application de différents noyaux lors de la classification des vecteurs de features. Nous avons essayé les noyaux : linéaire, polynomial, Laplacien, du chi-2 et RBF. Lors de nos différents essais avec les différentes features, il est apparu que le noyau du chi-2 donnait les meilleurs résultats.

Classification Ces noyaux ont été utilisés dans une classification SVM implémentée dans `svm.py`. Afin de réaliser une telle implémentation, nous nous sommes ramenés à la résolution d'un problème quadratique (QP) contraint de haute dimension ($(2d)^2$, où $d = 324$, le nombre final de features). La formulation générale de minimisation pour un SVM est donnée par :

$$\min_{f \in \mathcal{H}} \frac{1}{2}\|f\|_{\mathcal{H}}^2 + C \sum_{i=1}^{n} \varphi_{\textbf{hinge}}(y_i f(x_i))$$

avec pour dual :

$$\max_{\alpha \in \mathbb{R}^d} 2 \sum_{i=1}^{n} \alpha_i y_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

avec $0 \leq y_i \alpha_i \leq C$. Le majeur défaut de cette méthode est le temps nécessaire afin d'optimiser en plus du calcul de la matrice kernel. En commençant par un problème binaire, il fallait ensuite généraliser pour considérer toutes les classes, nous avons choisi d'utiliser une méthode one vs. one.

Bien que pour "one vs. one" il nous faille construire, entraîner et projeter à l'aide de $k(k-1)/2$ classificateurs, alors que pour "one vs. all" on ne nécessite que k classificateurs, il faut bien prendre en compte que les classificateurs n'agissent que sur une fraction des données dans le cas de "one vs. one". C'est pourquoi, il est difficile de juger de la complexité d'une stratégie comparée à l'autre. En termes d'efficacité, nous n'avons noté que peu de différences empiriques.

"Grid search" Pour trouver les meilleurs paramètres de marge C du SVM, gamma du noyau du chi-2 et les tailles et nombres des cellules des hogs, nous avons effectué une recherche dans une grille discrète de valeurs possibles des paramètres. Cela nous a permis de trouver les valeurs optimales suivantes :

- SVM - C = 10
- Chi 2 - Gamma = 3
- HOG °1 : 4 cellules de 8 pixels par blocs
- HOG °2 : 3 cellules de 10 pixels par blocs

"Data augmentation" (DA) L'échantillon de données était relativement petit pour entraîner des classificateurs robustes, avec une moyenne de seulement 500 images par classe. Nous avons alors fait appel à une technique très utilisée en Deep Learning et qui s'est révélée particulièrement efficace : l'augmentation de donnée. Cela consiste à générer des nouvelles images à partir de la base de données existantes en faisant subir aux images initiales de petites transformations aléatoires comme des rotations, des changements d'échelles. Les nouvelles images sont alors elles aussi utilisés pour l'entraînement du classificateur. Nous avons ainsi généré 40000 images supplémentaires en sélectionnant les transformations suivantes : une symétrie horizontale, une rotation maximale de 10 degrés, un décalage maximal horizontal et vertical de 0.2, un zoom entre $[0.8, 1.2]$, puis un angle de coupe de 0.2 radians au maximum. Cela a permis d'augmenter de presque 5% nos résultats initiaux, ce qui est un gain critique. Le nombre d'images est cependant limité par la mémoire de l'ordinateur (en particulier lors du calcul du kernel).

3 Résultats

Le tableau ci-dessous récapitule nos résultats sur les données d'entraînement et de test, selon les méthodes envisagées.

Pourcent. bien classés	Train	Test Int	Test Final
Filtres classiques	72	42.7	41.0
1 HOG, sans DA	81	57.3	55.4
1 HOG, avec DA	100	62.1	59.7
2 HOG, sans DA	100	58.9	56.6
2 HOG, avec DA	100	62.9	59.5

Les améliorations de score les plus significatives ont été obtenues en ajoutant successivement un HOG, puis en changeant le noyau pour un Chi2, et enfin en augmentant la base d'entraînement (DA). L'utilité de l'introduction du Double HOG est mitigée, puisque cette méthode est meilleure sur le jeu de test intermédiaire (Test Int), mais moins bonne sur le jeu de test final. Néanmoins, on note un problème global de sur-apprentissage, qui pourrait probablement être résolu en sélectionnant les features.