Object Detection with Discriminatively Trained Part Based Models (2010)

Théo Delemazure

January 2019

1 Introduction

Le but de l'algorithme présenté dans ce papier est de détecter des objets dans des photos (personnes, animaux, voitures). Pour cela, *Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester et Deva Ramanan* ont construit un modèle dans lequel les filtres sont passé à différentes résolutions de la photo, de façon à discriminer un objet selon son aspect général, puis selon ses différentes parties à plus haute résolution. Pour gérer la grande variabilité de forme des éléments (par exemple, voiture sous plusieurs angles), ils utilisent un *mixture model*.

2 Définition du modéle

2.1 Base du modéle

La base de cette méthode consiste à appliquer des filtres linéaires à des *features map* en calculant leur produit scalaire. Une *feature map* est une matrice de vecteurs qui permettent de représenter les zones d'une image selon certains paramètres (voir partie 5). On utilisera ici des pyramides de *feature maps* en calculant la *feature map* d'une image pour différentes résolutions. On note λ le nombre d'étages qu'il faut descendre dans la pyramide pour doubler la résolution de l'image.

Si $p = (x, y, l)$ est un point dans la *feature pyramid* H (où l est le niveau de résolution) et F un filtre on notera $F \cdot \Phi(H, p)$ le score de p pour F.

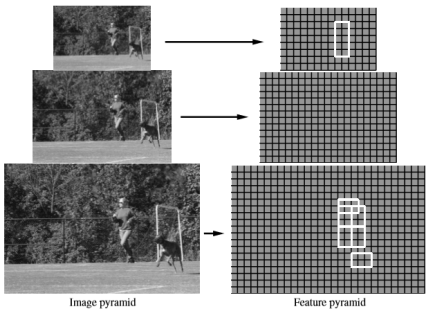

Figure 1 : *Feature pyramid*

2.2 Modéle à parties

La méthode est divisée en deux parties : tout d'abord, on détecte approximativement l'objet dans sa globalité avec un filtre grossier. Dans un second temps, on cherche les parties de cet objet avec des filtres plus précis (on prend une résolution deux fois plus importante dans la *feature pyramid*). Cela est illustré sur la **figure 1**.

On définit un modèle à n parties comme $M = (F_0, P_1, ..., P_N, b)$ où F_0 est le filtre racine pour l'image à basse résolution, b est le biais et $P_i = (F_i, v_i, d_i)$ correspond à la i^{eme} partie du modèle. Elle est caractérisée par son filtre F_i, sa position par rapport au filtre racine v_i et d_i est un quadruplet de coefficient quadratique utilisé dans les calculs.

Pour évaluer une hypothèse $z = (p_0, p_1, ..., p_n)$ (avec $\forall i > 0, l_i = l_0 - \lambda$) on utilise la formule suivante :

$$score(z) = \sum_{i=0}^{n} F_i \cdot \Phi(H, p_i) - \sum_{i=1}^{n} d_i \cdot \phi_d(dx_i, dy_i) + b$$

avec $(dx_i, dy_i) = (x_i, y_i) - (2(x_o, y_o) + v_i)$ le décalage et $\phi_d(a, b) = (a, b, a^2, b^2)$.

On note qu'on peut se ramener à un classificateur linéaire $\beta \cdot \Psi(H, z)$ avec $\beta = (F_0, ..., F_n, d_1, ..., d_n, b)$.

Le score d'une hypothèse d'un élément est la somme des hypothèses à partie :

$$score(p_0) = \max_{p_1, ...p, n} score(p_0, p_1, ..., p_n)$$

On calcule ce score avec de la programmation dynamique et une complexité raisonnable : En plus du calcul des filtres sur tous les points de l'image, la complexité est de $O(nk)$ où n est le nombres de parties et k le nombre d'étage dans la *feature pyramid*.

2.3 Modèle à plusieurs représentations

Un même objet peut avoir plusieurs modèles $(M_1, ..., M_m)$. On note une hypothèse $z = (c, z_c)$ où z_c est une hypothèse sur le modèle M_c. Ainsi on peut encore se ramener à un classificateur linéaire $\beta \cdot \Psi(H, z)$ avec $\beta = (\beta_1, ...\beta_m)$ et $\Psi(H, z) = (0, ..., 0, \Psi(H, z_c), 0, .., 0)$.

3 Entraîner un LSVM

On a vu que l'on a un classificateur que l'on veut optimiser et qui est de la forme

$$f_\beta(x) = \max_{z \in Z(x)} \beta \cdot \Phi(x, z)$$

où β correspond aux paramètres de notre modèle,x l'élément à évaluer (ici, une image) et z à des valeurs latentes (ici, une hypothèse).

On va entraîner β à minimiser une fonction de *loss*:

$$L(\beta) = \frac{\|\beta\|^2}{2} + C\sum_{i=1}^{n} max(0, 1 - y_i f_\beta(x_i))$$

Le problème de cette fonction de *loss* est qu'elle n'est pas convexe. Cependant, elle possède une propriété intéressante : **la semi-convexité**. En effet, si $y_i = -1$ la fonction $\beta \to max(0, 1 - y_i f_\beta(x_i)$ est convexe comme max de fonctions convexes. Ce n'est pas le cas si $y_i = 1$. Ainsi pour que L soit convexe, on suppose $y_i = 1 \Rightarrow |Z(x_i)| = 1$: puisqu'une seule valeur latente est possible, f_β est linéaire et donc $max(0, 1 - y_i f_\beta(x_i)$ est convexe.

On note alors Z_p l'ensemble des valeurs latentes choisie pour les exemples positifs et $L(\beta, Z_p)$ la *loss* associée. On effectue alors une descente coordonnée en deux étapes :

1. On optimise selon Z_p en choisissant les meilleurs hypothèses z_i : $z_i = argmax_{z \in Z(x_i)} \beta \cdot \Phi(x_i, z)$
2. On optimise selon β avec une descente de gradient classique : $\beta \leftarrow \beta - \alpha_t \times \nabla L_{D(Z_p)}(\beta)$, où α_t est le coefficient d'apprentissage au temps t (ici $\alpha_t = \frac{1}{t}$)

4 Exploiter les faux positifs

Notons $\beta^*(D) = argmin_\beta L_D(\beta)$ le β obtenu à l'issue de la descente de gradient, on voudrait trouver un sous-ensemble d'exemples $C \subset D$ tel que $\beta^*(C) = \beta^*(D)$. On divise les exemples en plusieurs ensembles et on s'intéresse aux *hard exemples* (faux positifs).

On fait alors une descente coordonnée utilisant un cache de faux positifs (P est un ensemble d'exemples positifs) :

1. On initialise le cache C_0 en y mettant des exemples choisis au hasard.
2. On fait une descente de gradient pour calculer le $\beta^*(C_t \cup P)$ où C_t est le cache actuel.
3. Si l'ensemble des faux positifs sous β est inclus dans le cache, on s'arrête.
4. Sinon, on supprime du cache les vrais positifs sous β, on y ajoute les faux positifs sous β qui ne sont pas dedans et on retourne à l'étape 2 avec C_{t+1}

On peut prouver que cet algorithme termine et que $\beta^*(C_{stable}) = \beta^*(D)$

5 Initialisation

5.1 Obtenir les *feature map*

Il existe différentes méthodes pour obtenir une *feature map* à partir d'une image. Dans notre cas, on utilise la norme et l'orientation discrétisée du gradient en tout point pour obtenir des 36-uplets. Pour faciliter les calculs et réduire la dimension, plusieurs méthodes sont proposées, par exemple réduire le nombre de valeurs à 11 en utilisant PCA.

5.2 Initialiser les paramètres

On initialise les filtres racines de façon intelligente à partir d'un entraînement sur les exemples positifs. On initialise les filtres des parties selon les zones "d'énergie" à l'intérieur du filtre racine et on suppose pour réduire le nombre de paramètre que les parties sont symétriques par rapport à l'axe médian.

6 Entrainement

L'algorithme final résume les parties 3 à 5 appliqués au modéle présenté en partie 2 :

1. On calcule l'ensemble optimal des exemples positifs (optimisation selon Z_p)
2. On fait la descente coordonnée présentée en partie 4 (optimisation selon β)

7 Optimisation

Trois optimisations sont proposées, permettant d'améliorer le score un petit peu:

- Prédiction de la position : On entraîne une fonction linéaire qui prend en argument $g(z)$ (vecteur de la position de la racine et de toutes les parties) et qui retourne la position et la taille de la *Bounding Box*.
- Suppression des valeur proches : Une fois qu'on a une liste de *Bounding Box* et leur score, on supprime les détections qui chevauchent 50% d'une détection à meilleur score.
- Prise en compte du contexte : La dernière optimisation consiste à entraîner selon les scores de chaque élément (voiture, chat, ...) pour choisir la classe de l'image.

8 Results

Dans la compétition de détection d'objet Pascal VOC 2008, cet algorithme a obtenu le meilleur score dans 9 des 20 catégories, dont les suivantes :

Category	bike	bird	bottle	person	plant
Score	0.402	0.117	0.284	0.431	0.117

Aujourd'hui, on obtient des scores dépassant 0.8 grâce aux CNN mais c'est un très bon résultat pour l'époque. Il a ouvert la voie à d'autres méthodes de détection, par exemple des CNN utilisant des modèles à parties.