Maxime DONNET
UTC GI04

24/05/2016

SY32 - Rapport de projet

Apprentissage supervisé et détection de visage sous Matlab

Table des matières

1) Introduction 2

2) Initialisation 2

3) Apprentissage 3

4) Nouvel Apprentissage 5

5) Détection de visage sur les images de test 7

6) Conclusion 9

1) Introduction

Le but de ce projet et de développer sous Matlab, un apprentissage supervisé afin d'obtenir un modèle de détection de visages de personnes sur des images de tests. Pour cela, nous disposons d'images d'entrainement fournis avec les coordonnées de la zone positive du visage. Il faut créer un modèle à partir des données positives, négative X et Y comme vu en cours et en TP récemment. Nous allons voir dans ce rapport, les différentes étapes pour créer un modèle et exploiter les données prédites afin d'obtenir un résultat de détection optimal.

2) Initialisation

Il faut commencer par charger les 1000 images d'entraînements fournis pour le TP. Il y a également 1000 labels associés contenant les coordonnées et hauteur, largeur du rectangle délimitant un visage. Je charge les images dans des cellules. Il y a une variable de 1000 cellules pour chaque image d'entrainements et une autre de 447 pour les images de test.

Il faut également trouver la plus petite dimension de visage afin de redimensionner l'ensemble des images d'entrainements par rapport à celle-ci. Cette valeur est de 46 ce qui correspond au maximum entre la largeur et hauteur du plus petit rectangle. Il nous a été conseillé, pour la détection de visage, de transformer les dimensions de label en carré car c'est plus simple à manipuler et cela s'accorde plutôt bien aux dimensions des visages humains. J'ai donc créé une nouvelle variable de 1000 lignes et 5 colonnes 'newlabel' qui contient le cadre des visages adapté aux images redimensionnées c'est-à-dire le X, Y, 46, 46 ainsi que la valeur de redimensionnement 'scale' avec laquelle il faut diviser la dimension de l'image pour qu'elle soit en accord avec celui du carré du visage. En effet, pour les procédures suivantes, le carré est fixe et c'est uniquement la taille de l'image qui est modifié pas à pas lors d'un glissage. Ce newlabel sera utilisé pour l'apprentissage et le nouvel apprentissage (prises en compte des faux positifs) sur les données d'entrainements. Pour toutes les procédures suivantes d'apprentissage et de détection, nous utilisons un carré de taille fixe 46 x 46 qui glisse sur une image.

Pour la suite du rapport, lorsque je parle de carré, je parle enfaite de l'image dans la zone délimitée par ce carré découpé sur une image de base.

3) Apprentissage

Pour l'apprentissage, qui consiste à créer un premier modèle de détection à partir des données d'entrainements, j'ai créé une fonction apprentissage 'apprentissage' qui prend comme argument l'ensemble des cellules d'images d'entrainements avec leurs nouveaux labels associés. Le but de cette fonction est de créée un modèle SVM sur des données positives et négatives et de récupérer les données d'entrainements X négatives et positives pour créer un nouveau modèle par la suite.

La 1ere partie de cette fonction consiste à récupérer les données positives, donc les visages sur chaque image. Je boucle ainsi sur les images d'entrainements 1000 fois. Il faut bien sur redimensionner l'image en divisant sa taille par le scale du newlabel. On récupère ensuite le visage avec la fonction 'imcrop' à partir des coordonnées de newlabel. Il suffit de redimensionner les valeurs des pixels du visage sur une ligne de notre variable d'entrainement positive X comme nous l'avions vu au préalable en séance de TP.

La 2eme partie de cette fonction consiste à récupérer les données négatives. Il a été indiqué de prendre 10 images négatives par images. On a donc une variable d'entrainements négatifs de 10000 lignes qui correspondent chacune à une image négative. Celle si sont des carré d'image de l'image d'entrainement pris au hasard. Pour maximiser le hasard, j'ai bouclé en changeant la dimension de l'image de base d'entrainement. Ainsi on récupère à chaque fois un carré au hasard sur l'image qui est également dimensionné de 1 à 1/6. Cependant, la difficulté consiste à ce que cette image ne soit pas trop proche de la zone positive.

Pour la première fonction je compare la différence en dur entre les coordonnées X et Y du carré positif avec celles du carré pris au hasard. Il faut qu'elles soient supérieures à une certaine valeur proportionnelle à la taille de l'image. Je compare également la différence entre le dimensionnement actuel de l'image sur laquelle le carré est pris avec le dimensionnement associé au carré positif. Il faut que cette différence soit inférieure à une certaine valeur car on pourrait très bien prendre une petite zone négative sur un visage et inversement avoir une grande zone négative contenant un visage (qui serait très petit). Après avoir normalisés les données d'entrainement X et Y, j'ai créé un premier modèle SVM avec un BoxConstraint = 2.^-5, valeur pris arbitrairement, ni trop grande ni trop petite, afin de rester précis sans que le modèle ne prennent trop de temps à se créer. Le modelé créé s'appelle 'modele1'.

Cependant j'ai remarqué que ce modèle n'était pas très précis. J'ai donc créée une deuxième fonction d'apprentissage 'apprentissage2' qui fait la même chose que 'apprentissage' sauf pour la condition qui permet d'enregistrer l'image pris au hasard dans la variable des données X négatives. J'ai créé en parallèle, une fonction 'intersection' qui permet de calculer le nombre de pixels en commun entre deux zones définies par deux rectangles (ou carrés). Elle utilise la fonction Matlab 'intersect' qui trouve les valeurs en communs entre deux tableaux. Ainsi, on peut maintenant considérer un carré pris au

hasard comme une donnée X négatives seulement si son intersection avec le carré positif (redimensionné à la dimension courante de l'image étudiée) retourne un résultat nul. Par soucis de précision, j'ai décidé de rejeter toutes les images qui pourraient recouvrir une partie ou faire partie du visage. J'ai également réajusté la BoxConstraint à 2.^-4 afin de calculer un modèle plus précis. Enfin, la fonction retourne également les données X positives et négatives, le modèle, mais également les images positives, les images négatives, et les images rejetés en tant que négatives dans des variables à cellules. Ainsi on pourra visualiser ces images pour vérifier qu'on a bien créé un modèle tel que nous le souhaitons. Le modèle créé s'appelle 'modele2'.

Le « Sliding-Window », consiste à faire glisser notre carré de dimension 46 x 46 sur une image afin de détecter par notre modèle si celui-ci correspond à un visage ou non.

Afin d'observer les résultats de mes modèles, j'ai créé une fonction glissage 'glissage' qui prend en argument une image, son label, le pas de glissage du carré, la dimension du carré et le modèle à appliquer. Cette fonction fait glisser un carré sur chaque ligne et colonne de l'image. Le carré glisse de pas en pas. Ainsi je récupère l'image délimitée par ce carré et je lui applique une prédiction avec le modèle SVM passé en argument. J'affiche le visage positif en jaune et tous les visages détectés (prediction==1) par le modèle en bleu.

J'ai pu remarquer qu'il existait des labels faux. En effet pour certaine images d'entrainements (une minorité), les coordonnées du visage fournis sont erronées. Cela va très probablement fausser le modèle et la détection des visages par celui-ci pour la suite.

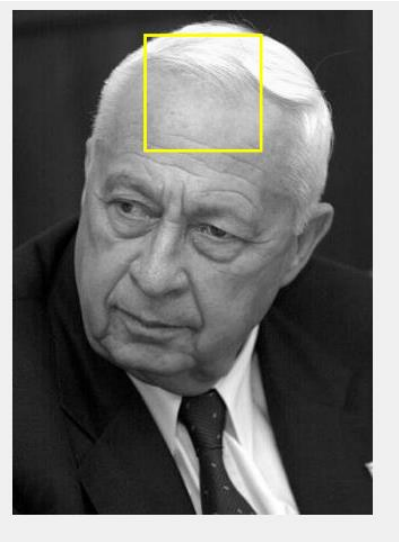

Figure 1: label de la zone positive erronés en jaune

J’ai pu constater que mes modèles trouvaient assez bien les visages lorsque l’image est dimensionné à la taille du Scale mais qu’il trouvait également pas mal de faux positifs, c’est-à-dire des visages là où il ne devrait pas y en avoir, notamment lorsque l’image est à grande taille. Cela est dû au fait que les modèles ne sont pas optimaux et qu’ils n’ont pas assez de données (positive et négatives) en leurs connaissance pour être réellement efficace. Pour la suite, je garde ‘modele2’ qui semblait plus performant dû à un meilleur apprentissage.

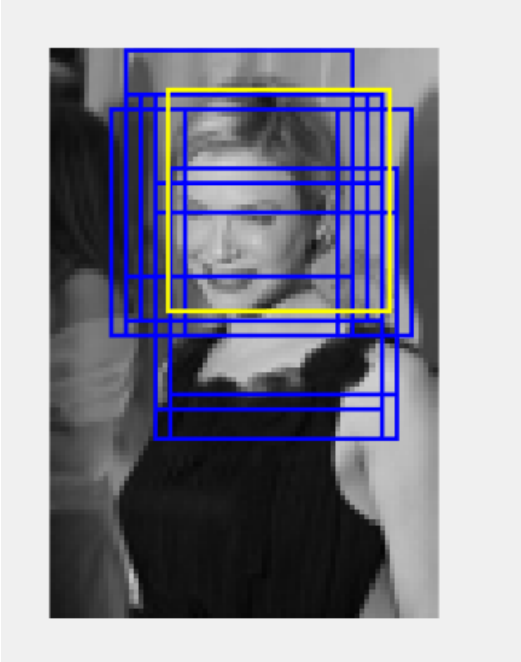

Figure 2: Résultat de 'glissage' avec trainC{97}, 'modele2', pas de 3 et scale de 3.5

4) Nouvel Apprentissage

Puisque le modèle ‘modele2’ n’était pas assez performant, j’ai décidé de l’améliorer en passant par l’étape du nouvel apprentissage. Le nouvel apprentissage consiste à récupérer tous les faux positifs pour les rajouter aux données négatives afin de recréer un nouveau modèle plus performant.

J’ai commencé par créer une nouvelle fonction ‘glissage2’ qui reprend ‘glissage’ à la différence qu’elle cherche et retourne les faux positifs et qu’elle redimensionne plusieurs fois l’image passé en paramètre. Elle va boucler en redimensionnant l’image de 1 à 1/10 avec un pas de 0.5 pour lui faire glisser le carré. Le pas de glissement de carré est proportionnel à la dimension courante de l’image. Lorsqu’il y a prédiction d’un visage, j’ai rajouté une condition pour vérifier si c’est un vrai ou un faux positif. Si l’intersection avec

le visage positif et l'image du carré courant n'est pas nulle, alors on la considère comme un vrai positif et affiche le contour du carré courant en bleu. Sinon, cela signifie que le visage prédit en tant que tel par le modèle passé en paramètre n'est pas à côté et n'a absolument rien à voir avec le vrai visage positif. On l'affiche en vert et on le considère donc comme un faux positif en l'ajoutant à la variable des nouvelles données négatives X du modèle.

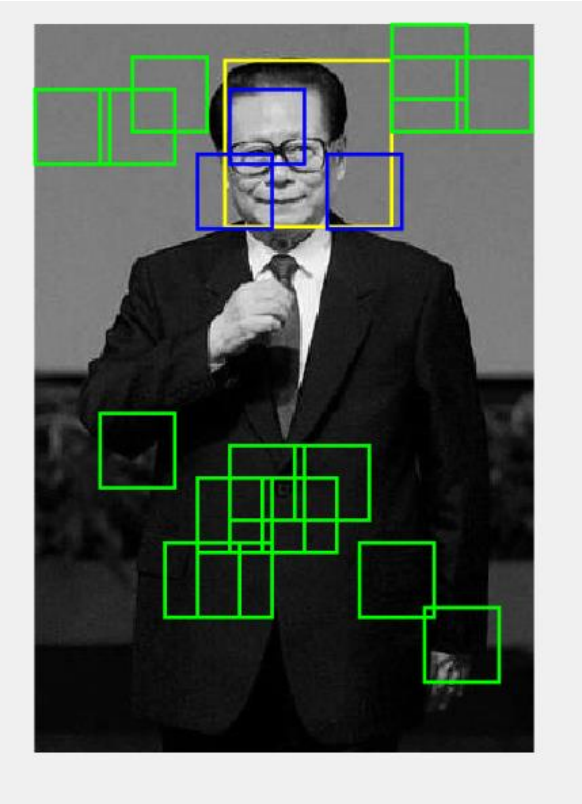

Figure 3: résultat de 'glissage2' avec trainC{700} et 'modele2'

Il faut maintenant prendre un échantillon des images d'entrainements et rajouter les faux positifs trouvés aux données d'entrainements négatives du modèle. Pour cela, j'ai créé la fonction 'nouveauApprentissage' qui prend en argument les images d'entrainements, leurs labels, le modèle, les données négatives et positives X de ce modèle et retourne le nouveau modèle créé. Elle boucle sur un échantillon d'images, récupère leurs faux positifs avec 'glissage2' et recrée un nouveau modèle SVM à partir des données positives passé en paramètre et les données faux positifs rajoutés à celle négatives passé en paramètre. J'ai donc bouclée sur 10 images, chaque centième de 1 à 1000, avec 'modele2' pour récupérer un nouveau modèle SVM de BoxConstraint = 2.^-4 'newModele'.

5) Détection de visage sur les images de test

Maintenant que j'ai 'newModele' censé être le modèle le plus performant, je vais pouvoir l'utiliser pour détecter les visages sur les images de test fournis pour le projet.

J'ai créé une nouvelle fonction appelé 'testImage' qui prend en argument une image, la dimension du carré et le modèle à appliquer et retourne l'ensemble des visages trouvés.

Cette fonction, de la même façon que 'glissage2', glisse le carré de pas en pas pour chaque dimension de l'image. On ajoute cette fois ci, lorsqu'il y a prédiction d'un visage, l'image délimité par le carré dans une variable de résultat. Lorsque toutes les dimensions et zones ont été parcourus, la deuxième partie de la fonction s'applique : éliminer les Non Maximas et afficher les visages. Cela consiste à étudier les carrés trouvés qui sont assez proche pour admettre qu'ils recouvrent le même visage et ne garder que celui avec le meilleur score. Pour cela, je vais parcourir toutes les images des carrés trouvés lors de la 1ere étape et les comparer entre eux. Il y a donc deux boucles imbriquées de 1 au nombre de visages trouvés. Si l'intersection entre deux images de carrés trouvés est supérieure à l'aire du carré de plus petite dimension alors cela signifie qu'ils détectent le même visage. En effet, il faut adapter l'aire au plus petit carré car si un petit carré se trouve dans la zone définie par un plus gros carré, cela signifie que celui-ci détecte en fait peut être un bout de visage ou alors inversement que le gros carré détecte le visage mais avec une vue plus large donc moins précise que le petit. Si cette condition est vérifiée, j'élimine le carré avec le moins bon score des deux. J'ai ainsi gardé uniquement les carrés prédisant un unique visage et pour lequel ils ont le meilleur score. A la fin de la fonction, l'image est affichée à sa dimension de base et les visages sont délimités par des carrés bleus. Je me charge également d'afficher en rouge le carré avec le meilleur score de tous, donc le visage le plus probable ou le mieux détecté de l'image.

Sur les deux images suivantes, les deux carrés affichés sont ceux encadrant deux visages distincts avec le meilleur score parmi les autres carrés qui encadraient le même visage. En rouge, le carré encadrant le visage le plus probable. On constate tout de même que le deuxième visage (en bleu) est une erreur.

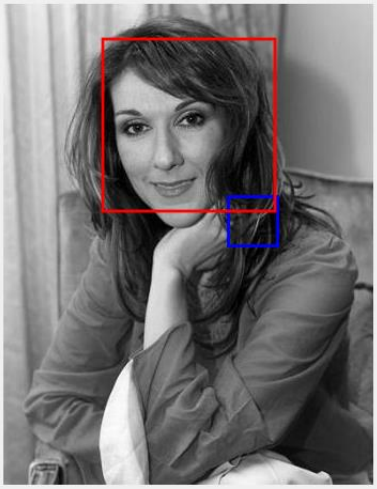

Figure 4: résultat de 'testImage' avec testC{104} et 'newModele'

Figure 5 : résultat de 'testImage' avec testC{254} et 'newModele'

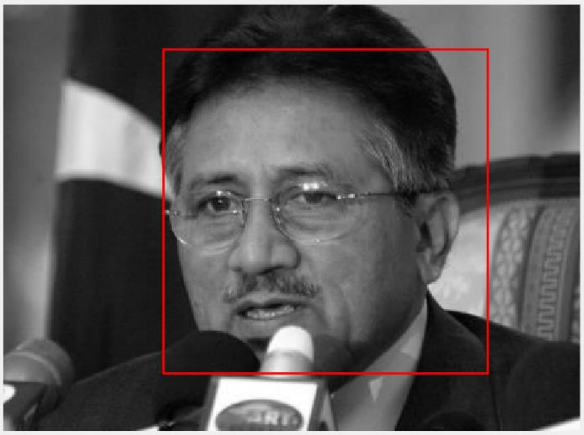

Figure 6:résultat de 'testImage' avec test{187} et 'newModele'

Enfin, j'ai créé la fonction 'detectionVisage' qui prend en argument l'ensemble des images sur lesquelles détecter des visages, la dimension du carré à faire glisser, et le modèle à appliquer. Cette fonction applique ainsi pour chaque image de test, la fonction 'testImage' afin de récupérer les coordonnées ainsi que le score de chaque visage détecté pour ensuite écrire ces mêmes informations dans un fichier .txt correspondant à l'image étudiée. On a donc tous les visages détectés par 'newModele' pour chacune des 447 images de test sous forme de fichier .txt regroupés dans un dossier.

6) Conclusion

J'ai malheureusement pu constater que si mon modèle trouvait des visages et parfois avec un bon score, ce n'était pas toujours le cas (faux négatif) et qu'il affichait encore des erreurs (faux positifs). Les algorithmes utilisés pour créer les modèles étaient pourtant correctes. Cela est surement du, comme nous l'avons vu avant, à la quantité des coordonnés des zones positive erronés donc de données positives erronés pour l'apprentissage qui faussent le modèle. Je pense qu'il faut également, afin d'avoir un modèle vraiment efficace et fonctionnel, disposer d'un nombre beaucoup plus grand de donnés positives et négatives et procéder à plusieurs réapprentissage avec les faux positifs. Cependant, j'ai constaté que sur Matlab, les boucles prennent énormément de temps et qu'il m'a fallu parfois plusieurs heures pour générer un modèle ou détecter des visages. J'ai tout de même appris à manipuler Matlab, à comprendre les étapes pour créer un modèle de détection et comment exploiter les données détectées afin de récupérer un résultat de détection optimal.