Arnaud DALIE P18

Rapport SY32 Groupe I

Introduction

L'objectif du projet est la construction d'un classifieur d'image par fenêtre glissante capable de détecter des visages à partir de photos.
Les librairies utilisées sont numPy, sklearn et skimage.

Les livrables attendus sont, en plus de ce rapport, l'ensemble des fichiers sources permettant de construire et d'évaluer le classifieur. Cet ensemble de fichier doit au moins contenir un script train.py et test.py respectivement chargés de l'entraînement et du test du classifieur.

Nous allons d'abord expliquer comment marchent train.py et test.py puis nous détaillerons les choix des procédés choisis

Fichier Train.py

Cette partie aura pour but de sommairement détailler le mode de fonctionnement de l'entraînement du classifieur.

Principe général

Le classifieur est entraîné en deux fois, une première fois sur un sous ensemble du dataset avec des exemples positifs fournis et des exemples négatifs généré aléatoirement, puis une seconde fois avec les mêmes positifs, mais également avec de faux négatifs issus du classifieur précédemment généré.

On a donc un procédé en 5 étapes : l'extraction des exemples positifs, la génération des exemples négatifs, l'entraînement du premier classifieur, la génération de nouveaux faux négatifs et enfin l'entraînement du nouveau classifieur.

Extraction des exemples positifs

Cette étape est l'une des plus simples. Grâce aux exemples et aux "labels" fournis, on peut extraire la sous-image de l'exemple contenant le visage. L'image extraite n'est quasiment jamais un carré alors que notre fenêtre glissante le sera, il faut donc remédier à ça. On ne peut pas utiliser la fonction resize qui pourrait, dans le cas d'une image top rectangulaire, aplatir l'image. Une première solution a été de prendre le carré inscrit dans le rectangle d'exemple mais cela supprimait le contexte autour du visage et le classifieur ne tenait alors pas compte des contours dans ses détections. Même problème pour le carré circonscrit au rectangle qui laissait au contraire trop d'importance au contexte environnant le visage.
La solution trouvée a donc été de faire une "moyenne" du carré inscrit et circonscrit.

```
positive = trainImg[
        (YSmall + YLarge) / 2: (YSmall + YLarge + dsmall + dLarge) / 2,
        (XSmall + XLarge) / 2: (XSmall + XLarge + dsmall + dLarge) / 2
        ]
```

Pour l'exemple généré, on calcule son image retournée et on ajoute les deux exemples aux positifs (nous passerons plus tard sur le hog appliqué à ces exemples)

Génération de négatifs

Les négatifs sont générés aléatoirement à partir des images fournis. On sélectionne au hasard des fenêtres et on les ajoutes aux négatifs. Afin d'être sûr de ne pas envoyer un visage dans les négatifs, on applique un filtre gaussien puissant sur les zones contenant les visages. Cette solution est loin d'être élégante et pourrait être remplacée par la non-prise en compte des images se trouvant dans les zones contenant des visages.

```
for j in range(negativeFactor) :
        #Correct if gaussian blur areas can be considered as FALSE
        ypos = random.randrange(1, len(trainImg) - slidingWindowSize)
        xpos = random.randrange(1, len(trainImg[0]) - slidingWindowSize)
        h = slidingWindowSize

        negImage = trainImg[ypos:ypos+h, xpos:xpos+h]
        fd = hog(negImage, pixels_per_cell=(8, 8), orientations=9)
        trainNegatives = np.append(trainNegatives, fd)
```

Entraînement du classifieur

Afin d'entraîner le classifieur, on utilise les positifs et négatifs précédemment générés. On attribue à ceux ci les valeurs True et False grâce à un vecteur Label. On concatène puis

mélange ces deux les labels et les exemples (positifs et négatifs) entre eux puis on utilise la fonction fit du classifieur de sklearn.

```
def trainClassifier(trainPositives, trainNegatives):
        labelPositive = np.ones(len(trainPositives), dtype = bool)
        labelNegative = np.zeros(len(trainNegatives), dtype = bool)

        trains = np.concatenate((trainPositives, trainNegatives))
        labels = np.concatenate((labelPositive, labelNegative))

        trains, labels = shuffle(trains, labels)

        clf = ABC()
        clf.fit(trains, labels)
        return clf
```

Génération des Négatifs à partir des faux Positifs

Cette partie est de loin la plus complexe et volumineuse du programme. Elle repose sur l'utilisation de la fonction testOneImage, elle même utilisant SlidingWindows et groupFaces.

Nous allons commencer par détailler ces fonctions avant d'expliquer brièvement comment elles s'articulent entre elles.

La fonction SlidingWindow

Cette fonction prend simplement en argument la hauteur et la largeur de l'image à étudier. On place une sous-fenêtre dont les dimensions sont précisées dans config.py aux coordonnées (0,0). Ensuite on fait avancer la fenêtre sur l'axe x de la taille de la fenêtre multiplié par le stepFactor (lui aussi précisé dans config.py). Ainsi une fenêtre de 30 pixels avec un stepFactor de 0.3 avancera de 10 pixels par 10 pixels. Quand on arrive au bout d'une ligne on se place sur la ligne (+ 10 pixels sur l'axe y) suivante et on recommence.
A chaque itération on ajoute les coordonnées x et y et la largeur de la boîte qui sont retournée dès qu'on a atteint la fin du fichier.

```
while (x + slidingWindowSize < l - 1 or y + slidingWindowSize < h - 1):

        box = [x, y, slidingWindowSize]
        AllBoxes = np.append(AllBoxes, box)

        #Computation of new coordinates

        #if we are at the end of a row, we go to the next row
        if (x + slidingWindowSize > l - 1) :
        x = 0
```

```
y = y + slidingWindowSize * stepFactor
#In all other cases, we go to the next box
else :
x = x + slidingWindowSize * stepFactor
```

La fonction testOneImage

Cette fonction commence par lancer slidingWindow et pour chaque tuple retournée, extrait la sous image de l'image passée en argument et prédit si cette sous image est un visage. Si tel est le cas, les coordonnées testées sont ajoutées à la liste retournée par testOneImage.
On répète cette opération avec une image réduite en taille afin d'artificiellement augmenter la taille de la fenêtre glissante. Cette opération est répétée tant que la fenêtre glissante est plus petite que l'image testée.

Avant de retourner la liste finale des coordonnées, on applique la fonction groupFace qui permet de supprimer de la liste les coordonnées décrivant des fenêtre se superposant sur au moins la moitié leur aire d'union

```
while(h > slidingWindowSize and l > slidingWindowSize):
print(h)
print(l)
AllBoxes = slidingWindow(h, l)
print(len(AllBoxes))

for i in AllBoxes:
b += 1
box = img[
        int(i[1]): int(i[1])+int(i[2]),
        int(i[0]): int(i[0])+int(i[2])
        ]
#Magic trick to deal with side collisions of my sliding window
blackBox = np.zeros((slidingWindowSize,slidingWindowSize))
blackBox[0: len(box), 0:len(box[0])] = box

box = blackBox

fd = hog(box, pixels_per_cell=config.Cell, orientations=9)

if (clf.predict(fd.reshape(1, -1))):

        #allow to get true boxes and not only 32 pixel boxes with fake dims
        for j in range(len(i)):
        i[j] = i[j] * accelerator * (1/scaleFactor)**(trueSizeFactor - 1)

        facesDetected = np.append(facesDetected, i)
```

```
                facesDetected = np.append(facesDetected,
clf.decision_function(fd.reshape(1, -1))[0])

        h = h * scaleFactor
        l = l * scaleFactor
        img = resize(img, (int(h), int(l)))
        trueSizeFactor += 1
```

La fonction groupFace

Cette fonction est calcule les superposition en comparant une à une les coordonnées fournies dans la liste. Cette fonction est donc très coûteuse vu qu'elle s'applique à chaque image testée. Pour comparer les coordonnées, on applique la fonction whichBoxToRemove avec les deux coordonnées à comparer. Cette fonction renverra 0 si il n'y a rien à supprimer, 1 si on doit supprimer le premier argument et 2 pour le second.

Pour savoir laquelle des deux supprimer, on vérifie d'abord qu'elles sont en contact. Si oui, on calcule l'aire du rectangle d'intersection des deux fenêtres afin d'en déduire :
Aire(union) / Aire(Inter).

```
left = max(x1, x2)
right = min(x1 + h1, x2 + h2)
bottom = min(y1 + h1, y2 + h2)
top = max(y1, y2)

H = bottom - top
L = right - left
if (H < 0 or L < 0):
        return 0


A1 = h1 ** 2
A2 = h2 ** 2

inter = H * L

union = A1 + A2 - inter
Area = inter / union
```

Si cette aire dépasse le seuil de ½ on renvoie "l'indice" de la fenêtre à laquelle le classifieur à attribué le plus haut score. Cette boîte est ensuite supprimée de la liste des coordonnées retournées.

La génération des exemples négatifs à partir des faux Positifs

Pour chaque image restante dans le training set, on appelle testOneImage qui renvoie toutes les fenêtres contenant un visage selon le classifieur. On va comparer, de la même manière que dans groupFace, les fenêtre trouvée à la fenêtre réelle. Si 0 est renvoyé, c'est

qu'elle sont de faux positifs et elles sont donc ajoutées au faux Positifs et aux exemples négatifs en vu de l'entraînement du second classifieur.

Le fichier Test.py

Ce fichier permet de tester le classifieur obtenu précédemment mais cette fois-ci sur l'ensemble de test dont on ne connaît pas les résultats. Tout cela est fait par la même fonction que précédemment, testOneImage. Cette fois ci, les résultats sont ajoutés au fichier result.txt et cheat.txt. result.txt contient toutes les fenêtres renvoyées alors que cheat.txt ne contient que la fenêtre la plus probable.

Le choix des paramètres

Cette partie a pour but d'expliquer les choix faits pour traiter les images en amont de tout entraînement. Cela inclut l'utilisation d'un hog, le choix du classifieur ou encore de la taille des fenêtres glissantes. Ces choix sont pour la plupart des variables globales déclarées dans config.py

Utilisation du hog

Le classifieur ne fait des calculs que sur des hog. Ceux-ci sont calculés avant chaque ajout aux vecteurs d'exemples positifs et négatifs et avant chaque prédiction par le classifieur.
Bien que le Hog apporte une solide description des formes de l'image, l'utilisation de celui-ci est particulièrement coûteux et dépendant du nombre de pixels par cellule. Plus cette valeure se réduit, plus la taille des données envoyées au classifieur augmente.

SVM ou Adaboost ?

Bien que la solution AdaBoost ait été envisagé au début du projet, c'est un SVM qui a été utilisé avec un noyau rbf, réputé plus performant pour le traitement d'image. L'AdaBoost avait tendance à se focaliser sur les grosses variations de luminosité et donc à classer comme positif des lettres dans le fond de l'image ou des costumes (noir sur blanc).
Au vu du nombre important de prédiction négatives que le classifieur doit effectuer (comparé aux positives) ; et de la qualité des exemples positifs (aucun n'est inexact), il fallait un très grand C pour le classifieur. Cela permet de générer un hyperplan qui classera bien un maximum des exemples du jeu d'entraînement. Ce grand C, fixé grossièrement à 100.000, exige donc une grande qualité des exemples négatifs générés, ce qui est une des faiblesses du projet.

Taille de la fenêtre glissante

La valeur choisi pour la taille de la fenêtre glissante est de 36 pixels. Cela apporte un bon équilibre entre niveau de détail et temps de calcul.
Deux autres paramètres liés à la fenêtre glissante sont stepFactor et scaleFactor qui décrivent l'augmentation relative de taille de la fenêtre glissante et le décalage de celle-ci à chaque itération de l'algorithme de test.

Accelerator

Étant limité par la puissance de calcul des machines utilisées, il a fallu réduire artificiellement le nombre de fenêtre pour lesquelles les hog sont calculés. Les images faisant toutes environ 450 pixels sur 450 et la taille des fenêtres glissantes étant de 36, il y aurait beaucoup trop de sous fenêtre à traiter (10 mn pour la prédiction d'une image pour plus de 40000 fenêtre étudiées).
Pour pallier à ce problème, on divise la taille de l'image par la valeur de accelerator avant de la tester. Cela a bien évidemment pour conséquence de réduire les performances sur les photos contenant de "petit visages".

```
h = len(img) / accelerator
l = len(img[0]) / accelerator

img = resize(img, (int(h), int(l)))
```

Paramètres sur la génération des négatifs

3 paramètres sont dédiés à la génération d'exemples négatifs.

AreaTresh

L'idée derrière ce paramètre est que lors de la génération de la deuxième moitié des négatifs, il arrive de trouver des bouts de visages qui ne remplissent pas la condition de superposition car la fenêtre est trop petite par rapport à la fenêtre attendue. L'idée est d'adapter ce seuil non plus à un demi comme de rigueur, mais à une valeure arbitraire permettant de ne pas ajouter un "gros" bout de visage dans les négatifs tout en continuant d'y ajouter les trops petites detections

NegativeFactor

Ce paramètre correspond tout simplement au nombre d'exemples négatifs générés pour un exemple positif. A noter qu'il faut essayer d'équilibrer le nombre de négatifs aléatoire par rapport au faux positifs de la deuxième phase afin qu'un groupe ne prenne pas le dessus sur l'autre.

TrainingFactor

Ce paramètre correspond à la proportion d'image utilisées pour la deuxième phase d'entraînement. J'ai pu lire sur des forums que 60% - 40% était dans les valeurs optimales.

Résultats et Conclusion

Ce projet aura été l'occasion de découvrir le sujet passionnant et parfois polémique qu'est la vision par ordinateur et plus particulièrement la détection de visage, le tout sous un angle technique.
Les méthodes utilisées, comme les fenêtres glissantes ou l'entraînement en deux phases, permettent d'arriver à de bon résultats qui peuvent tout de même être améliorée. Dans la majeure partie des images, la fenêtre avec le plus haut score correspond bien à un visage.
De nombreuses améliorations sont envisageables. L'optimisation est l'amélioration la plus évidente et sans doute la plus inaccessible. Une autre piste plus accessible serait l'amélioration des exemples négatifs générés. En effet, l'application du flou n'apporte pas grand chose aux exemples négatifs puisqu'il détruit les gradients de l'image. Il pourrait être intéressant de mettre en place un système de "metrics" permettant de faire varier "intelligemment" les différents paramètre de l'algorithme. Enfin, une dernière optimisation concerne le bug sur le fichier texte généré, qui ne se génère jamais complètement et s'arrête quelques images avant la fin alors que l'algorithme a tourné sur toutes les photos.