SY32 Détection de visage

Yan LIU et Xiang LI

19 avril 2019

Table des matières

1 Introduction

L'objectif du projet est la construction d'un classifieur robuste d'image par la méthode de fenêtre glissante capable de détecter des visages à partir de photos. Nous avons deux sources à notre disposition :

— 1000 images

— Une liste de coordonnées qui nous aide à repérer les visages sur ces images

A partir de ces deux sources, nous avons entraîné notre classifieur et définit des fonctions pour repérer un visage, évaluer un résultat... Dans notre projet, les librairies utilisées sont : numpy, skimage, os, random, sklearn, matplotlib. La réalisation se compose de cinq processus principaux :

— Extraction des exemples positives

— Génération des exemples négatives

— Entraînement du classifieur

— Détection des visages via la fenêtre glissante

— Apprentissage d'un nouveau classifieur avec des exemples faux positives

2 Réalisation

2.1 Extraction des exemples positives

C'est la première étape du projet, qui est la plus facile. Dans la liste de "label", chaque ligne représente le numéro de l'image, la ligne, la colonne, la hauteur et la largeur. Grâce à ces informations, nous avons pu découper la partie du visage. Dans cette étape, nous n'avons pas traité les images fournies, après la coupe, nous les avons sauvegardés directement.

Extraction

```
cropped = x_train[index][
                      int(x_label[i][1]):int(x_label[i][1]+x_label[i][3]),
                      int(x_label[i][2]):int(x_label[i][2]+x_label[i][4])
                      ]
```

2.2 Génération des exemples négatives

Nombre d'images

La génération des exemples est aléatoire. Pour chaque image, nous avons décidé de générer 10 exemples en considérant la vitesse et la performance. Parce que plus d'exemples

que nous générons, plus robuste le classifieur serait, cependant, ce fait est au prix du temps d'apprentissage. Ici, dix est un compromis.

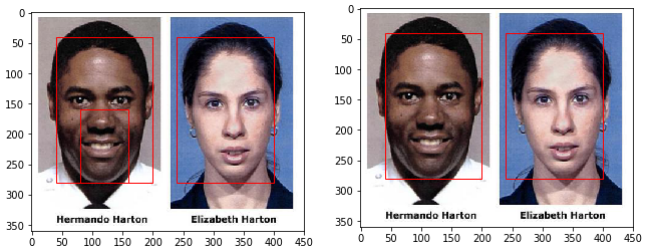
Visage ou non ?

En raison que la génération est aléatoire, nous sommes risque de générer d'une image contenant le visage. Pour éviter ce problème, nous avons défini une fonction qui permet d'évaluer et vérifier si l'image générée est valide. Le critère est en fonction du ratio de l'aire JOIN et l'aire UNION. Précisément, si la partie confondue de l'image générée et de la partie du visage occupe une moitié d'aire sur leurs aires totales tronquée la partie confondue, nous la rejette. Cependant, il est possible d'avoir plusieurs visages dans une image. Nous avons donc comparé l'image fournie avec tous les visages dans l'image originale.

isVisage

```
def aireJoinUnion(rc1x, rc1y, rc1L, rc1H, rc2x, rc2y, rc2L, rc2H):
    p1x = max(rc1x, rc2x)
    p1y = max(rc1y, rc2y)
    p2x = min(rc1x + rc1L, rc2x + rc2L)
    p2y = min(rc1y + rc1H, rc2y + rc2H)
    AJoin = 0
    if p2x >= p1x and p2y >= p1y:
        AJoin = (p2x - p1x)*(p2y - p1y)
    else:
        return 0
    A1 = rc1L*rc1H
    A2 = rc1L*rc2H
    if AJoin/A1 > 0.8 or AJoin/A2 > 0.8:
        return 1
    AUnion = A1 + A2 - AJoin
    return AJoin/AUnion
```

En revenche, cette méthode ne peut pas gérer le cas où un rectangle est trop petit et l'autre est grand. Par conséquent, nous avons ajouté le critère que la zone de coïncidence doit être inférieure à 80% de tout les deux rectangles.



Taille de l'image

Pour la taille des images négatives, nous avons décidé de prendre la taille du visage comme le choix, c'est-à-dire, la hauteur et la largeur dans "label". S'il y a plusieurs "labels" pour une image, nous prenons compte la taille moyenne.

Toutefois, nous avons éventuellement rencontré un cas que le visage occupe trop de parties dans l'image originale, une photographie de profil, ce que nous rend impossible de générer une image négative ayant la même taille que ce visage (toujours non valide). Dans ce cas, nous avons réduit la hauteur et la largeur jusque nous puissions générer assez d'images négatives valides.

ispHeadPortrait

```
def ispHeadPortrait(im,L, H):
    isHeadPortrait = False
    l = im.shape[1]
    h = im.shape[0]
    s_im = l * h
    s_extract = L * H
    if (s_extract/s_im ) > 0.4:
        isHeadPortrait = True
    return isHeadPortrait
```

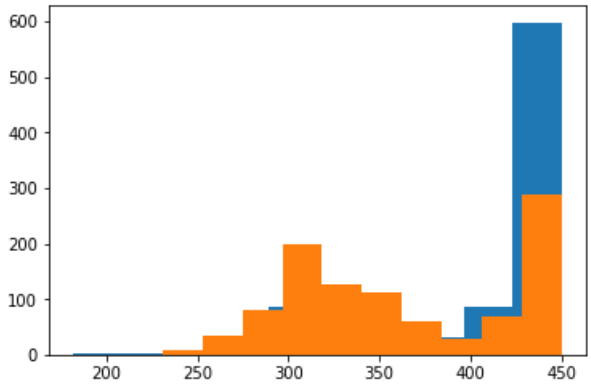
2.3 Entraînement du classifieur

Choix de la taille de boîte

Afin d'entraîner le classifieur, nous avons utilisés les positifs et négatifs générés précédemment. Pour l'apprentissage, toutes les images doivent être la même taille. Pour ce faire, nous avons défini une fonction, qui nous aide à rendre les images en degré gris, et les redimensionner. Concernant la taille à redimensionner, nous avons pris une hauteur et une largeur ayant 1,5 comme le ratio. Cette valeur est la moyenne du ratio Hauteur/-Largeur de tous les visages.

```
In [13]: np.mean(x_label[:,3]/x_label[:,4])
Out[13]: 1.519537040270022
```

Et puis nous avons fait un histogramme de la distribution des hauteurs et des largeurs des visages trouver une bonne taille de boîte englobante. D'après le ratio et la distribution, on a choisi 48 comme hauteur est 32 comme largeur.



Feature

Dans la pratique, nous avons calculé le HOG pour chaque image. Les paramètres de HOG par défaut ont été utilisée. Le classifieur fourni est beaucoup plus performant que celui-ci entraîné avec seulement des pixels. Le calcul de HOG s'est fait juste avant de mettre dans le modèle d'apprentissage, car nous avons fait aucun traitement lors de l'étape "extraction des positives" et "génération des négatives". Parce que nous avons pensé garder plus d'informations, pour la situation que nous pensons manipuler un autre descripteur sans remanipuler l'étape de l'extraction et la génération.

Méthode d'apprentissage

On a choisi SVM comme la méthode d'apprentissage. Pour mieux utiliser les sources à l'apprentissage on a aussi utilisé CrossValidation.

```
def CrossValidation(xtrain, ytrain, Nvc):
    clf = svm.SVC(kernel='linear')
    r = np.zeros(Nvc)
    for i in range(Nvc):
        print("boucle␣:␣%d"%(i))
        mask = np.zeros(xtrain.shape[0], dtype = bool)
        mask[np.arange(i, mask.size, Nvc)] = True
        clf.fit(xtrain[~mask], ytrain[~mask])
        r[i] = np.mean(clf.predict(xtrain[mask]) != ytrain[mask])
    print("Taux␣d'erreur␣:␣%d"%(np.mean(r)))
    return clf
```

Evaluation de méthode

L'évaluation d'une méthode est toujours commencée en comparant avec le classifieur naturel. Dans notre cas, nous avons eu 1284 exemples positives et 10000 négatives. Alors, si on prédit le résultat tous en négatif, alors le taux d'erreur est $1284/11284 \approx 11,4\%$

```
dataTrain = trainPos + trainNeg
dataTrain = np.ravel(dataTrain).reshape(len(trainNeg) + len(trainPos)
    ↪ ,4860)
clf = CrossValidation(dataTrain, yTrain , 5)
yPred = clf.predict(dataTrain)
print(f"Taux␣d'erreurs:␣{np.mean(yPred␣!=␣yTrain)*100}")
```

Notre classifieur a eu $0,43\%$ comme le taux d'erreur. C'est un bon résultat par rapport au classifieur naturel.

2.4 Fenêtre glissante

Taille de boîte

La valeur choisie pour la taille de la fenêtre glissante est identique que celle-ci des exemples positives et négatives redimensionnées. Du fait, la prédiction est plus facile. Il nous fallait juste transmettre la partie cadrée par la boîte en gris et puis calculer son HOG sans se soucier le problème de la taille. Chaque itération, la fenêtre déplace une distance fixée, qu'on dit un pas. La taille du pas est aussi un paramètre à considérer. Il influence le temps que la fenêtre parcourir toute l'image. En considérant l'efficacité et la précision, nous avons pris un quart de largeur de boîte comme la taille de pas. C'est un choix le plus équilibre.

Choix de l'échelle

Dans la pratique, nous avons manipulé la stratégie que changer l'échelle de la fenêtre. L'échelle qu'on a prise est :

```
scale = [0.5,0.4,0.3,0.2,0.1]
```

Nous avons abandonné les échelles de 1 à 0,6. Parce qu'au plupart du cas, les images sont grandes, on peut faire une hypothèse qu'il y a peu de visages trop petits. Cependant notre fenêtre est petite, si on garde l'image inchangée, le temps de détection augmente, le bruit augmente également(faux positives).

Vérification de visage

Pour vérifier si c'est un visage ou pas, on utilise clf.predict ainsi que son score. S'il est bien un visage (dont score est supérieur à un seuil), on garde ses coordonnées est son score dans deux liste.

```
carrepredict = clf.predict(np.ravel(carre).reshape(1,4860))
             score = clf.decision_function(np.ravel(carre).reshape
                 ↪ (1,4860))
if(carrepredict[0] == 1 and score > 0.8 ):
       listxy.append((np.array([x, y, resL, resH]).astype(int)))
```

```
        listscore.append(np.array(score))
```

Suppression des superpostions

La fonction précédente, fenetreglissante, nous fournit deux listes : coordonnées et scores d'images. Donc on peut supprimer les superpositions via la fonction aireJoinUnion et la liste de score. Et puis retourner deux nouvelles listes.

```
def nms(coords, scores):
    i_base = 0
    while i_base < len(coords):
        i_new = 0
        while i_new < len(coords):
            ratioAireJoinUnion = aireJoinUnion(coords[i_base][0],coords[
                ↪ i_base][1],coords[i_base][2],coords[i_base][3],coords[
                ↪ i_new][0],coords[i_new][1],coords[i_new][2],coords[i_new
                ↪ ][3])
            if ratioAireJoinUnion > 0.5:
                if scores[i_new][0] > scores[i_base][0]:
                    del coords[i_base]
                    del scores[i_base]
                    i_base -= 1
                    break
                elif scores[i_new][0] < scores[i_base][0]:
                    del coords[i_new]
                    del scores[i_new]
                    i_new -= 1
            i_new += 1
        i_base += 1
    return coords, scores
```

Dessiner le cadre

Jusqu'à maintenant, on a une liste de coordonnées des visages détectés. A l'aide de fonction **patches.Rectangle** dans la librairie matplotlib, nous avons pu dessiner les cadres sur l'image. Cette action nous rend plus intuitif le résultat obtenu.

```
fig,ax = plt.subplots(1)
ax.imshow(x_train[0])
for i in range(len(coord)):
    x = coord[i][0]
    y = coord[i][1]
    L = coord[i][2]
    H = coord[i][3]
```

```
#(x,y),width,height
rect = patches.Rectangle((x,y),L,H,linewidth=1,edgecolor='r',facecolor
    ↪ ='none')
ax.add_patch(rect)
```

2.5 Faux Positives

Pour nous, la précision est au prioritaire, nous n'avons pas envie d'avoir tort de se tromper n'importe quoi comme un visage. Pour ce fait, nous avons parcouru le jeu de données original via la fenêtre glissante, et fait comparer avec le visage positive. Nous avons pris les faux positives comme les négatives, et puis re-entraîné notre classifieur pour lui rend plus robuste.

Dans pratique, nous avons affecté les visages positives un score 99999, qui est forcément supérieur que les autres scores. Ainsi, après la suppression de superposition, si une image ayant le score inférieur à 99999, elle sera un exemple faux positive.

Cette fois-ci, le taux d'erreur est quasiment zéro.

2.6 Choix du seuil

Dans la fonction de fenêtre glissante, nous avons mis un seuil. La taille de fenêtre, l'échelle, le classifieur, tous ont une influence sur le choix de seuil. Par exemple, le nouveau classifieur est plus strict que l'ancien, nous avons ainsi baissé le seuil pour avoir encore une bonne "rappel". Cependant la façon du choix du score n'est pas très élégante, nous n'avons pas dessiné le courbe de ROC complètement. Nous avons appliqué notre classifieur sur un sous-ensemble d'images, et vu les résultats à l'œil nu pour comparer quel seuil semble meilleur. Finalement, nous avons pris 0,45 comme le seuil.

3 Conclusion et Améliorations possibles

Pour des raisons de manque de temps, nous n'avons pas pu tenter toutes les méthodes qu'on a appris. Nous avons pensé des améliorations possibles.

Amélioration des paramètres

Dans l'apprentissage, nous avons effectué cross-validation pour choisir un bon kit de paramètre. Cependant, nous pourrions encore tester avec différents paramètres : la taille de pas, la taille de boîte, l'échelle du zoom, etc.

Choix de feature

A part la descripteur HOG, nous avons vu également la dense DAISY et les caractéristiques pseudo-Haar. Parmi ces deux méthode, pseudo-Harr est très simples et très rapides à calculer, surtout à l'aide des images intégrales. La dense DAISY est similaire à SIFT, qui génère l'histogramme du gradient pour chaque bloc, cependant, la stratégie pour décomposer en bloc suit la loi Gaussien. Tous ces deux descripteurs nous intéressent.

Apprentissage profonde

L'apprentissage profonde peut augmenter effectivement la précision. Par rapport aux méthodes classiques, il nous permettra de profiter de beaucoup plus de données sans "gaspiller" pour un bon résultat. Si on a eu plus énorme de données, cette méthode était au prioritaire.

Conclusion

Finalement, nous avons réussi à obtenir un programme à reconnaissance de visage. La précision est autour de 92%, cependant le rappel est seulement 72%. Ce résultat accorde notre prévu, parce qu'on souligne la précision lors que le rappel reste un niveau acceptable. Ce projet nous a permis d'être familial avec la méthode de fenêtre glissante et mettre en pratique les théories au niveau d'apprentissage et la détection d'objet.

4 Annexe

train.py

Dans ce script, nous avons mis toutes les fonctions et instructions permettant de générer le classifieur. Les adresses de sources et de sortie sont attendues à saisir selon votre cas. Et le processus de l'apprentissage en ajoutant les faux positives est identique que celui-ci ancien, nous n'avons donc pas remis le code de réapprentissage à la fin.

test.py

Vous pouvez faire tourner ce script sans pre-exécuter **train.py**.

```
def sortie(txt_path, x_test):
    f = open(txt_path,'a')
    for i in range(len(x_test)):
        print(i)
        result = fenetreglissante(util.img_as_float(color.rgb2gray(x_test[
            ↪ i])))
        coord = result[0]
        score = result[1]

        coord = nms(coord,score)[0]
        score = nms(coord,score)[1]
        for j in range(len(coord)):
            x = coord[j][0]
            y = coord[j][1]
            L = coord[j][2]
            H = coord[j][3]
            f.writelines(['\n', "%03d"%(i+1) ,'␣',str(y), '␣', str(x), '␣',
                ↪  str(H), '␣', str(L), '␣', str( float("{0:.2f}".format(
                ↪ score[j][0])) ) ])
    f.close()
```

txt_path, **x_test** représentent respectivement l'adresse complète du fichier de sortie et les images à détecter les visages.

Voilà un exemple d'utilisation :

```
s= 'E:\GI05\SY32\SY32_projet\data\detection.txt'
xtst = load_images_from_folder('E:\GI05\SY32\SY32_projet\data\\test')
sortie(s, xtst)
```