Université de Technologie de Compiègne

SY32

Rapport Projet

Détection de visages

Printemps 2019

BRASSEUR Alexandre - PENNORS Josselin

19 avril 2019

1 Introduction

Ce projet a pour but de créer un algorithme de vision par ordinateur permettant de détecter des visages sur des images. L'objectif est d'établir le modèle le plus efficace et ainsi détecter sur un ensemble test le plus de visages possibles tout en ne détectant pas comme étant un visage un contenu qui n'en serait pas un. Bien que largement utilisé de nos jours, aucun algorithme d'apprentissage profond n'a été utilisé. Nous avons en effet utilisé les classifieurs de la bibliothèque Scikit-Learn. De plus pour toutes nos manipulations nous avons eu recours aux bibliothèques Numpy et Scikit-Image.

Pour nous permettre d'avancer sur le projet, nous avons utilisé Gitlab UTC pour stocker, versionner et travailler de paire sur ce projet.

2 Structure générale

Pour réaliser ce projet nous avons dans un premier temps pris le temps d'organiser les principales tâches à effectuer avant d'être en mesure de pouvoir proposer le modèle le plus performant pour détecter des visages. Ainsi nous avons identifié différentes fonctionnalités à réaliser :

- Chargement des images représentant un visage et découpage pour avoir un jeu de données d'entraînement et un jeu de données de validation ;
- Génération d'un jeu de données d'images ne représentant pas de visage ;
- Entraînement d'un modèle ;
- Enregistrement d'un modèle dans un fichier ;
- Chargement d'un modèle à partir d'un fichier ;
- Prédiction sur les images du jeu de données initial via la méthode des fenêtres glissantes ;
- Identification des fenêtres prédites comme possédant un visage mais n'étant pas validées comme telles ;
- Nouvel entraînement du modèle avec les données précédemment utilisées ainsi que les fenêtres identifiées au point précédant et appelées "fausses positives" ;
- Utilisation du jeu de données de validation pour prédire toutes les fenêtres possédant un visage de par la méthode des fenêtres glissantes et pour estimer l'erreur du modèle en le validant ;
- Génération d'une courbe précision/rappel grâce à la validation des images de validation ;
- Enregitrement de toutes les statistiques des prédictions du modèle dans un fichier (vrai positif, faux positif, vrai négatif, faux négatif) ;
- Généralisation du processus pour essayer plusieurs modèles avec différents paramètres.

Pour des soucis d'organisation nous avons créé différents fichiers Python. Il y a tout d'abord deux fichiers principaux à la racine du projet qui sont *train.py* et *test.py* qui comme leurs noms l'indiquent ont respectivement pour mission d'un côté d'entraîner le modèle et le stocker dans un ficher et de l'autre de charger le modèle à partir du fichier et de réaliser les prédictions sur le jeu de données de validation. Ces deux fichiers sont paramétrés par le fichier *config.py* où se trouvent la configuration de notre détecteur. Enfin nous avons créé un dossier *modules* contenant les différents composants nous ayant permis de réaliser le projet, à savoir :

- *data.py* pour la manipulation des données (chargement, sauvegarde) ;
- *descriptor_vector.py* pour la conversion des images en vecteur descripteur ;
- *models.py* pour toutes les fonctions liées au modèle ;
- *negative_set.py* pour la génération du jeu de données ne possédant pas de visages ;
- *selection.py* pour réaliser des tests sur les différents modèles ;

— *utils.py* pour les outils divers tels que le calcul de l'aire de recouvrement entre deux fenêtres ;

— *validation.py* pour valider les prédictions et retourner les différentes statistiques associées ;

— *window.py* pour réaliser la méthode des fenêtres glissantes.

3 Données d'entrée du modèle

Avant de pouvoir commencer à entraîner un modèle, la première étape importante a été de prendre en main les données d'entrées. Nous disposions en effet d'un jeu de données d'entraînement composé de 1000 images de différentes tailles ainsi que d'un fichier *.txt* comportant les emplacements de tous les visages du jeu de données. Chaque emplacement est de la forme suivante : Numéro d'image, x, y, h, l. Nous disposions également d'un jeu de données de test composé de 500 images qui nous sera utile plus tard dans le projet pour tester l'efficacité de notre détecteur.

Ce projet revient à un problème de classification binaire. Pour une boîte englobante donnée, cette dernière représente un visage ou pas. C'est pourquoi nous avons dans un premier temps concentré nos efforts à la création de deux classes, l'une représentant exclusivement des visages et l'autre non.

3.1 Jeu de données positif

Pour constituer le jeu de données positif, c'est à dire le jeu de données représentant des visages, nous avons utilisé le fichier .txt qui nous était fourni.

Ce dernier contient en effet les labels de tous les visages présentes dans les images de la forme Numéro d'image, x, y, h, l. Nous avons alors pu aisément itérer dans les labels et récupérer les fenêtres des images associées grâce aux coordonées. De plus, lors de la prise en main de ce jeu de données, nous avons regardé le ratio moyen entre la hauteur d'un visage et sa largeur, et nous avons trouvé une valeur de 1.6. Cette donnée nous a été utile par la suite du projet dans le but d'avoir un format similaire pour les images manipulées, notamment lors de changement de taille qui pourrait entraîner une déformation imporante.

Enfin ce jeu de données a été divisé en 2, avec 70% de jeu de données utilisé pour l'apprentissage et le reste pour la validation de nos modèles.

3.2 Génération du jeu de données négatif

Les images ayant été découpées pour récupérer les fenêtres représentant des visages, il a fallu ensuite générer des images ne représentant pas de visages. Pour cela nous avons utilisé les mêmes images que le jeu de données d'entraînement.

Nous avons ainsi créer une fonction pour générer ce jeu de données. Cette dernière prend ainsi en paramètre les images, les labels des visages et un paramètre fixant la taille du jeu de données. Nous avons gardé ce paramètre comme une variable dans le fichier principal que nous pouvions aisément modifier.

Pour générer une image ne comportant pas de visage, nous avons tout d'abord fixé une hauteur minimum de 60px. Nous avons par la suite générer aléatoirement une hauteur et une largeur de sorte qu'il y ait le même ratio, défini dans la partie précédente, que pour les visages. Ainsi, la taille des images est variable, mais le ratio est toujours le même. En

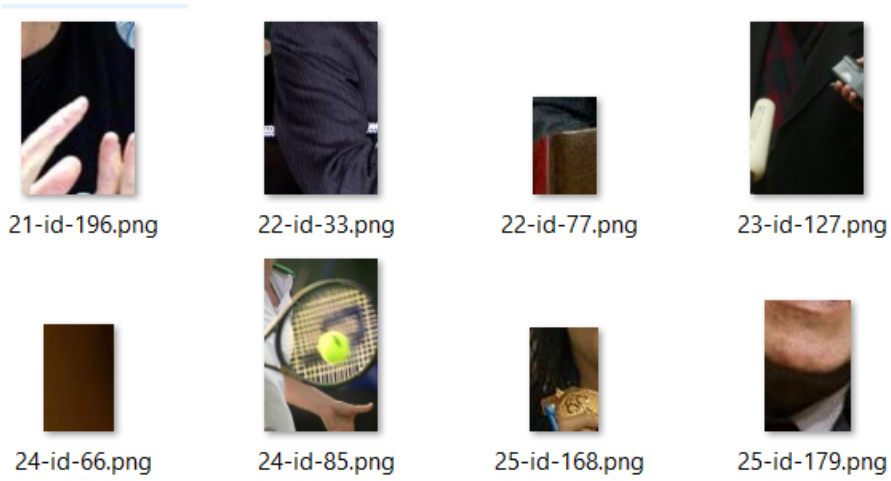
se basant sur la hauteur et la largeur nous avons ensuite pu générer la position x et y de l'image.

Cette dernière étant générée aléatoirement en position et en échelle et ne devant pas comporté de visage, une fonction a par la suite été créée pour vérifier quelle ne représentait bien pas de visage. Pour se faire, nous avons récupéré le numéro de l'image utilisé pour la créer. Grâce à ce numéro et aux labels, il a été possible de vérifier pour tous les visages présents sur l'image que l'aire de recouvrement entre ces derniers et la nouvelle fenêtre n'était pas supérieur à 0.5. Dans le cas contraire, nous avons considéré que l'image représentait un visage et que par conséquent il ne devait pas être utilisé dans le jeu de données négatif.

L'algorithme itère jusqu'à trouver le nombre d'images sans visage envoyé en paramètre de la fonction principale.

3.3 Enregistrement des images

Lorsque nous avons commencé à programmer ce projet, nous avons rencontré de nombreuses difficultés pour réaliser les différents fonctionnalités fixés initialement. Le manque de clarté dans les données manipulées, qui ne sont autre que des tableaux de valeurs, nous a poussé à enregistrer les images dans des dossiers pour les visualiser. Cela a notamment été le cas pour la génération du jeu de données ne représentant pas de visages. Il est préférable de vérifier tant que possible que les fenêtres créées par notre algorithme ne correspondent pas à des visages. Voici un exemple des images enregistrées :



Ainsi nous avons pu suivre l'évolution des images utilisées pour entraîner le modèle dans les jeux de données positif et négatif.

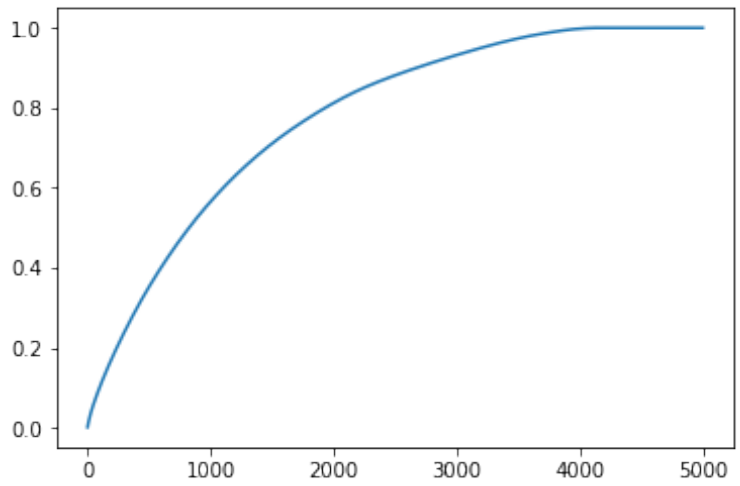
3.4 Création des vecteurs descripteurs

Une fois les images récupérées à un ratio constant, nous avons procédé à une transformation dans une taille fixe, à savoir une hauteur de 40px et une largeur de 25px. Cette taille s'explique par le fait qu'elle correspond environ au ratio trouvé lors de la prise en main des données et que nous ne pouvons utiliser un format plus grand en raison de calculs qui deviennent trop lourds par la suite.

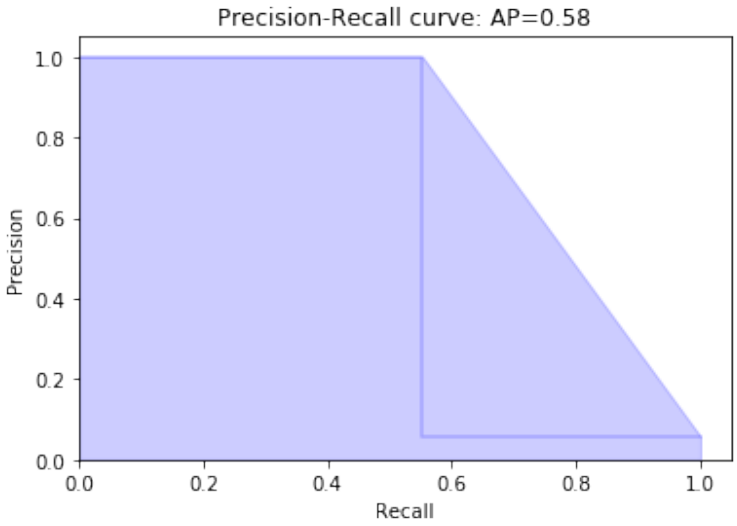
Nous les avons ensuite transformer en vecteurs descipteurs pour en extraire des caractéristiques. Les images sont aussi transformées en niveaux de gris, car nous avons observé que la couleur importait peu, de plus cela facilite les calcule pour certaines transformations. Nous avons aussi redimensionner les morceaux d'images avant de les transformer, cela permet de garder une dimension de sortie constante nécessaire à l'apprentissage.

Nous avons vu en cours plusieurs méthodes pour décrire une image telle que *HOG*, l'Histogramme des Gradients Orientés, *DAISY*, une technique de transformation de charactérstiques invariante à l'échelle, et les charactéristiques pseudo-Haar. Lors de la plupart de nos essais, nous nous sommes concentrés sur l'usage de HOG qui est simple à prendre en main, rapide et avait été performant lors du TD sur les caractéristiques visuelles.

L'utilisation des charactéristiques pseudo-Haar était assez difficile. Pour un morceaux d'images de 40 par 25 pixels, nous obtenons un vecteur contenant 81900 valeurs. Ce qui alourdissait énormément les calculs. Nous avons donc essayé de diminuer cette taille en sélectionnant les coordonnées de ce vecteur qui avaient le plus d'importance. L'importance du vecteur est disponible via le classifieur entraînée par *clf.feature_importances_* . Après avoir trié ce vecteur d'importance et réalisé la somme cumulative, nous avons obtenu les résultats suivants :



Ainsi les 4040 meilleurs coordonnées réalisaient $99,9\%$ de la classification. Nous avons donc utilisé ces coordonnées et entraîné un nouveau classifieur avec, réduisant ainsi la taille des vecteurs par 20. Cependant nous avons obtenu des résultats bien moindre à l'utilisation d'un même modèle avec HOG, cela pourrait être dû à une erreur lors du traitement.


Precision-Recall curve: AP=0.58
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

4 Entraînement d'un modèle

Nous avons réalisé l'entraînement du modèle en deux phases afin de permettre un meilleur apprentissage, tout en veillant à ne pas sur-apprendre.

4.1 Première phase

Dans un premier temps, grâce aux données obtenues dans la précédente partie il a été possible d'entraîner un modèle en utilisant les vecteurs descripteurs et les labels de classe (+1 pour les vecteurs descripteurs de visages et -1 pour les autres).

Pour terminer cette première phase d'apprentissage, nous avons réutilisé toutes les images du jeu de données d'entraînement pour faire des prédictions. Ainsi de par un module Python que nous avons créé nous avons réalisé la méthode des fenêtres glissantes. Pour cette dernière toutes les images ont une à une été parcourues selon trois axes : leur échelle qui a été variée, leur axe x et leur axe y. Cela a permis de parcourir les images sous tous les points de vue possibles et d'avoir des visages en gros plan, des visages coupés, des plans d'ensemble et des éléments n'ayant rien à voir avec des visages. Cela a généré de nombreuses fenêtres par image. C'est pourquoi le module a été appelé pour les images une à une et les prédictions étaient faites, de par les vecteurs descripteur de ces dernières, au fur et à mesure des itérations pour ne pas avoir de saturation de mémoire. Nous avons utilisé des boîtes englobantes de 150px de hauteur et 90px de largeur. Les images ont été réduites de 30px de hauteur de largeur à chaque itération. Les coordonnées renvoyées étaient celles du repère d'origine et non celles dans l'image redimensionnée pour garder une cohérence générale des données.

Ainsi pour toutes les images obtenues par la méthode des fenêtres glissantes, une prédiction était faite, permettant d'obtenir un score. Ce dernier sert à savoir si le modèle prédit un visage ou pas. Le but de ce premier entraînement n'étant pas d'évaluer le modèle mais de permettre un entraînement supplémentaire, nous avons ainsi récupéré toutes les classifications dites "faux positifs", c'est à dire toutes les fenêtres pour lesquelles l'algorithme a prédit un visage alors que ce n'en était pas un.

4.2 Deuxième phase

La deuxième phase d'apprentissage démarre avec les données utilisés pour la première phase et avec les "faux positifs" obtenus lors de la validation de cette même phase. Le modèle est ainsi entraîné et doit être plus efficace que le précédent car les fenêtres que ce dernier aurait associé à des visages ont été placés dans le jeu de données négatif. Ce dernier peut ainsi être validé avec le jeu de données adéquat.

5 Prédiction sur les données de validation

5.1 Prédiction

Après avoir réalisé les deux phases d'apprentissage, les réelles prédiction peuvent commencer sur le jeu de données de validation. De la même manière que pendant l'apprentissage, via la méthode des fenêtres glissantes nous avons pu prédire la classe de nombreuses fenêtres couvrant chaque image sous chaque point de vue.

5.2 Validation

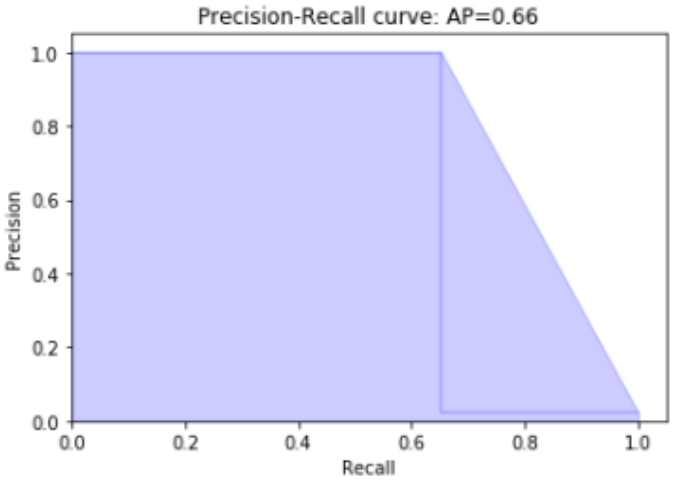
Les prédictions permettent d'obtenir un score nous indiquant pour chaque fenêtre la classe que le modèle a choisi, à savoir si cette fenêtre représente un visage ou pas. En prenant les labels fournis dans le projet il est possible de vérifier si une prédiction positive est représente réellement à un visage et si au contraire une prédiction négative ne correspond pas à un visage. Grâce à cela il est possible d'obtenir 4 statistiques essentielles pour comprendre l'efficacité du modèle : les "vrais positifs", les "faux positifs", les "vrais négatifs" et les "faux négatifs". A chaque prédiction, une de ses classes est ainsi incrémentée.

5.3 Courbe Précision/Rappel

Lors de la validation et de l'incrémentation des différentes catégories de prédiction, la courbe de Précision/Rappel peut être construite. A chaque itération, en prenant en compte le nombre de "vrais positifs", de "faux positifs" et de "faux négatifs" il est possible de calculer la précision et le rappel. La précision peut se définir comme le taux de bonnes prédictions du modèle parmis les prédictions positives (prédiction d'une image d'un visage). Le rappel peut se définir comme le taux d'images de visage reconnus par les prédictions parmis le nombre total de visages.

Les scores des prédictions sont triés par ordre décroissant, allant ainsi de la plus grande probabilité selon le modèle de la présence d'un visage jusqu'à la plus petite probabilité. A chaque itération, un point peut être ajouté à la courbe grâce aux totaux calculés. Voici par exemple une courbe Précision/Rappel réalisé avec le classifieur Random Forest sur un petit échantillon.

```
# Trying classifier `random_forest`...
Vectorizing data...
First training with (3308, 243) rows...
Predicting windows 100% 300/300 [00:49<00:00, 5.34it/s]
Adding 6 false positives / 51 predictions
Vectorizing data...
Second training with (3314, 243) rows...
```



6 Choix des paramètres

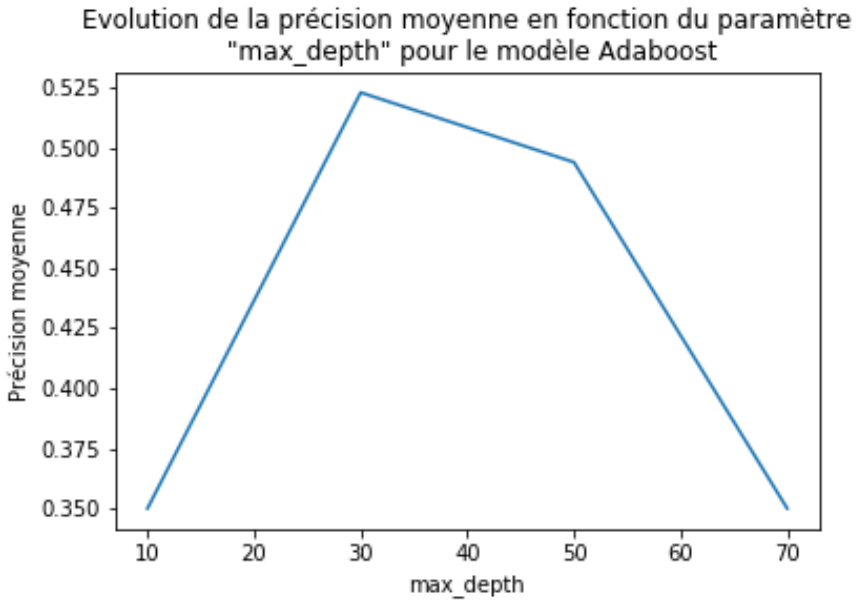
Nous avons lors de ce projet essayé différents classifieurs avec différents paramètres. Pour cela nous avons mis au point un module et un notebook pour effectuer divers tests sur les différents classifieurs.Nous avons aussi créer des fonctions nous permettant d'effectuer de multiples tests d'un coup, facilitant le choix de modèles et de paramètres.

6.1 Choix du classifieur

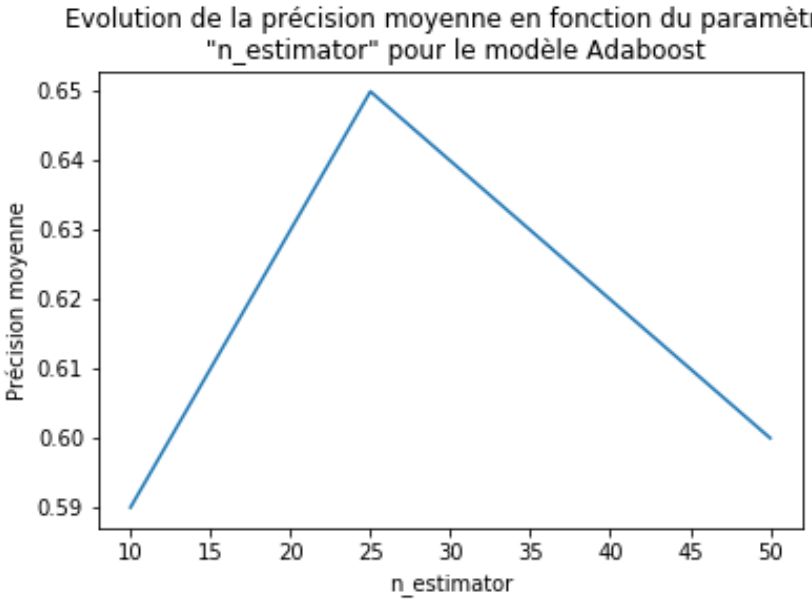
Afin de prédire la position des visages sur une image, il a fallu tout d'abord réussir à les détecter. Ainsi, à partir d'un morceau d'image, le rôle du classifieur a été de décider ou non s'il s'agit d'un visage.

6.1.1 Adaboost

Pour le modèle Adaboost, le module *AdaBoostClassifier* de bibliothèque Scikit-Learn a été utilisé. Deux paramètres pouvant faire varier les performances du modèle ont été identifiés à savoir « n_estimator » et « max_depth ». Ainsi lors d'essai sur différentes valeurs de « max_depth » nous avons obtenu cette courbe :



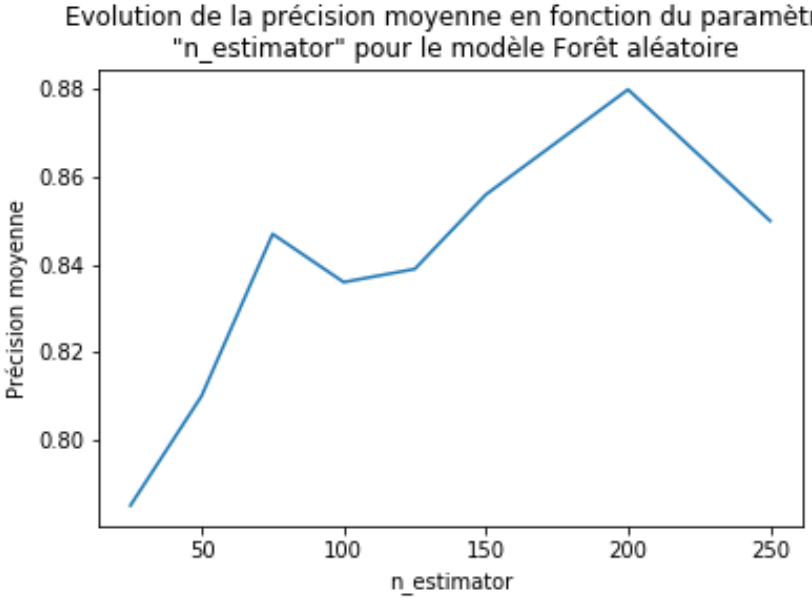
Nous avons ainsi décidé de retenir la valeur 30 pour ce paramètre et de faire des tests sur le paramètre « n_estimator ». La courbe suivante a été obtenue :



Contenu des faibles scores nous avons décidé de mettre de côté ce modèle qui n'a pas semblé être adapté à notre projet.

6.1.2 Forêt aléatoire

Nous avons par la suite essayé d'utiliser le module *RandomForestClassifier* de la bibliothèque Scikit-Learn. Le principal paramètre pouvant être modifié dans ce modèle était le nombre d'arbre dans la forêt appelé "n_estimator". Nous avons fait varié ce nombre afin de voir quel en était la valeur optimale. Les résultats suivants ont été trouvés sur le jeu de données de validation.



Ainsi la précision moyenne semble être élevée, le nombre optimal d'arbre semble tourner autour de 200.

La forêt aléatoire étant une généralisation de l'arbre de décision nous avons pris la décision de ne pas tester ce dernier.

6.1.3 Linear SVC

Pour ce modèle, nous avons modifié un paramètre à savoir le paramètre C. En le faisant varier nous avons obtenu des résultats assez similaires qui tournaient autour d'une précision moyenne de 0,9. Ce modèle avait ainsi des résultats intéressants.

6.2 Génération des fenêtres glissantes

Une fois le classifieur choisi, il faut générer des fenêtres glissantes sur toute l'image afin de détecter les visages. Pour cela nous avons créé une fonction qui prend en compte deux paramètres principaux : les dimensions de la fenêtre et le pas de déplacement. De plus, pour chaque image nous la redimensionnement en plus petit, afin de vérifier chaque échelle possible. Cela ajoute le paramètre du pas de redimensionnement.

Pour la taille de la fenêtre, nous avons regardé les dimensions moyennes des visages et pris une valeur légèrement inférieure (100 par 65 pixels) afin d'optimiser l'utilisation du redimensionnement d'image.

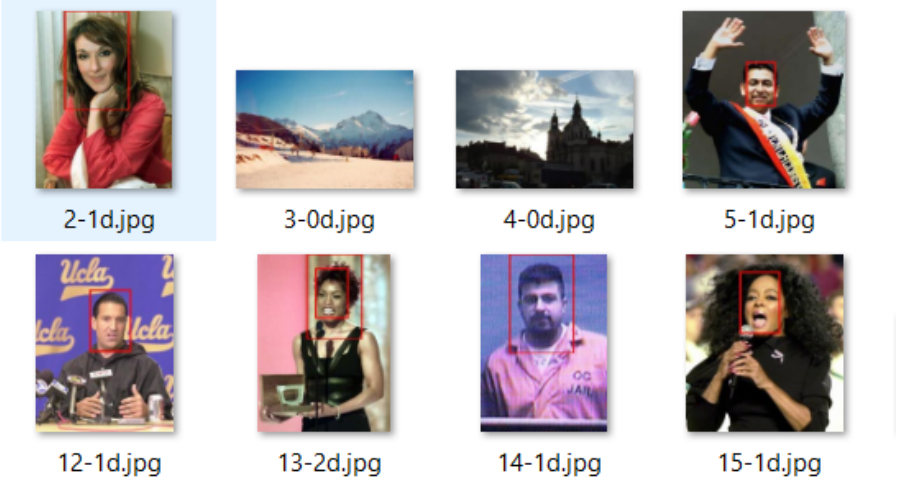
Pour le pas de déplacement, 30 par 30 pixels semblait optimal et présentait un bon compromis entre le parcours totale de l'image et la limitation de déplacement inutile.

Pour le pas de redimensionnement, 30 pixels réalise aussi un bon compromis entre le nombre d'images générées et les différentes échelles parcourues.

6.3 Résultats sur le jeu de données test

Nous avons ainsi réalisé des essais avec les modèles Linear SVC et Random Forest sur le jeu de données de test afin d'obtenir un fichier .txt des prédictions sous la forme (numéro_img, x, y, h, l, score). Ce fichier servira par la suite pour évaluer la performance du modèle. C'est ainsi que nous avons pu utiliser les 3 essais accordés en ligne pour tester notre modèle.

Nous avons également à une évaluation visuelle de notre modèle. Ainsi après génération du fichier des prédictions, nous avons un notebook créé une fonction pour récupérer toutes les images, ajouter les cadres prédits par notre modèle et sauvegarder ces dernières dans un nouveau dossier. Cela nous a permis de regarder si les coordonnées trouvées étaient cohérentes. Voici un exemple de résultats :



6.4 Choix du modèle

Après avoir testé les différents modèles évoqués, avoir réalisé des tests via le site et avoir visualisé les images de test avec les cadres des prédictions, nous avons gardé le choix du modèle Forêt aléatoire avec 200 comme valeur du paramètre n_estimator.

7 Comment utiliser notre projet ?

7.1 Données d'entrée

A la racine du projet, un dossier *data* a été créé. Nous y avons mis dedans le fichier *label.txt*. Deux dossiers sont présents également, à savoir *train* dans lequel il faut ranger toutes les images d'apprentissage et *test* dans lequel il faut ranger toutes les images de test.

7.2 Simulation de l'entaînement et de la prédiction

Il est possible d'exécuter le fichier *train.py* à la racine du projet. Ce dernier va s'occuper d'entraîner le modèle à partir des visages et du jeu de données négatif généré. Ce modèle sera enregistré par la suite. Il faut ensuite exécuter le fichier *test.py* qui permettra de charger le modèle et de faire des prédictions sur les images de test après génération de diverses fenêtres. Les fenêtres par le modèle comme représentant un visage sont enregistrées dans le fichier *detection.txt*. Tous les paramètres permettant le processus sont enregistrés dans le fichier *config.py*

8 Conclusion

Ainsi, ce projet a été très enrichissant. Il nous a permis de développer de nombreuses connaissances, telles que notre maîtrise de Python et des bibliothèques que ce dernier offre (Numpy, Sckit-Learn, Scikit-Image). Il nous a également permis de manipuler les différents classifieurs vus en cours avec un objectif concret et visualisable.

Nous avons rencontré de nombreuses difficultés dans la construction de ce projet. Les données représentent un point essentiel et il est très important de garder une cohérence générale dans leur format pour permettre les connexions entre les nombreux modules. Ce projet nous a fait prendre conscience de l'importance du traitement de la donnée, notamment lors de manipulation de grands volumes. Il a également été essentiel d'optimiser les traitements de ne pas dupliquer inutilement des données telles qu'une image. La manipulation de tels jeux de données peut prendre du temps et il est essentiel d'organiser au mieux son rendement (test sur des petits jeux de données, utilisation de notebook).