Description du projet

Ici, nous allons simplement déterminer si au moins un masque est porté dans chacune des photos passées en paramètre. Nous allons donc classifier l'image donnée en paramètre dans une des deux classes suivantes :

- **1 : au moins un masque est porté dans la photo**
- **0 : aucun masque n'est porté dans la photo**

Description de l'algorithme

Premièrement, nous allons contituer notre dataset d'images dans le but d'entrainer par la suite l'algorothme de notre modele.
Pour cela, nous nous baser sur 2 banques d'images libres.

- **Premier dataset fourni par Kaggle : Face Mask Detection**

Elle contient 853 images de personnes portant un masque et de personnes n'en portant pas. Chacune d'elles est associée à un fichier xml détaillant les caractéristiques du port du masque des personnes présentes sur la photo.

Ce dernier comporte une balise `<object>` pour chaque personne présente sur la photo, detaillant via plusieurs balises filles les informations sur la personne :
`<name>` indique si un masque est porté par la personne, et de quelle manière.
`<bndbox>` décrit une "boite" situant le visage de la personne sur la photo via ses coordonnées dans les balises `<xmin>, <ymin>, <xmax>` et `<ymax>`

```
<object>
        <name>with_mask</name>
        <pose>           </pose>
        <truncated> </truncated>
        <occluded> </occluded>
        <difficult> </difficult>
        <bndbox>
            <xmin>98</xmin>
            <ymin>62</ymin>
            <xmax>114</xmax>
            <ymax>80</ymax>
        </bndbox>
</object>
```

```
<object>
        <name>mask_weared_incorrect</name>
        <pose>           </pose>
        <truncated> </truncated>
        <occluded> </occluded>
        <difficult> </difficult>
        <bndbox>
            <xmin>162</xmin>
            <ymin>51</ymin>
            <xmax>194</xmax>
            <ymax>89</ymax>
        </bndbox>
</object>
```

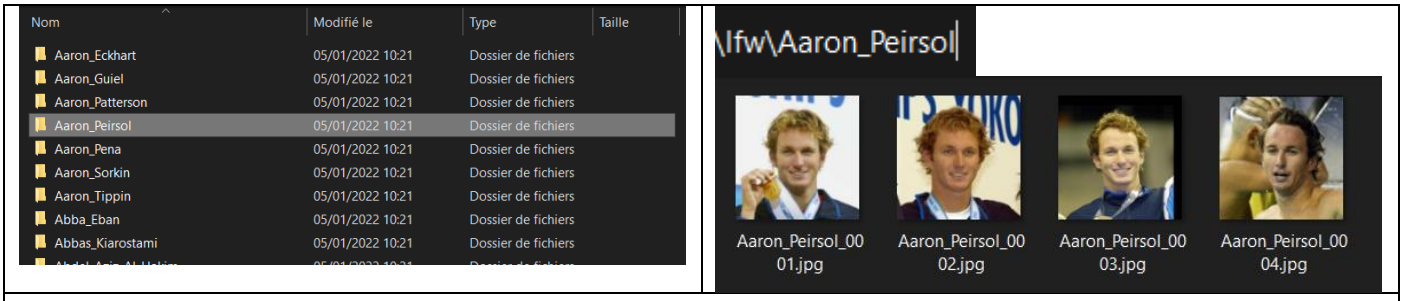
Cependant, comme indiqué dans la partie "Description du projet", nous cherchons ici seulement déterminer la présence ou non d'au moins un masque porté dans les photos analysées. Ainsi, nous choisissons de considérer comme valide et présent un masque incorrectement porté dans une photo
Notes :

Nous nous étions initialement basé uniquement sur ce dataset pour entrainer notre model. Cepandant, bien que les test effectuée sur une partie isolé de la banque d'image soient concluant, il était difficile pour notre modele de reconnaitre de nouvelles images de personnes ne portant aucun masque. Apres analyse plus approfondie du dataset, il nous a semblait que ce dernier ne présentait pas suffisamment d'image de personnes sans masques pour pouvoir s'entrainer de manière efficace.
Ainsi, nous avons choisi de faire appel à une autre base de données ne contenant cette fois ci que des images de personnes ne portant pas de masque.

- **Deuxième dataset fourni par Kaggle : Labelled Faces in the Wild (LFW) Dataset**

Ce deuxième dataset est extrait de LFW, un ensemble conséquent de datasets à destination des algorithmes de vision par ordinateur. Il contient 2854 photos de personnes seules.
De plus il s'organise différemment du précèdent. En effet, dans le but de pouvoir par exemple l'utiliser pour réaliser

un programme de reconnaissance de l'identité des personnes présentes sur une photo, les images sont groupées par nom et prénoms dans une hiérarchie de sous-répertoires.



Chaque sous-repertoire du dataset comporte une ou plusieurs images liées à une même personne

Malgré tout, nous n'allons là non plus pas utiliser cette spécificité dans le cadre de notre projet et charger simplement les images de cette base, indépendant de l'identité de la personne présente.

Au total, notre modèle sera entrainé sur un dataset de 2854 photos

Chargement des datasets

Nous souhaitons alors charger les images de ces 2 datasets au sein de notre programme Python.
Nous choisissons pour cela de les stocker dans un dictionnaire 'data', destiné à contenir l'ensemble de nos données d'entrainement et de test du modèle. Il comportera 3 champs :
- le champs de clé `'description'`, contenant une simple description de son contenu.
- le champs de clé `'data'`, une liste de toutes les images chargées sous forme de tableau numpy.
- le champs de clé `'label'`, une liste contenant le nom des classes ('0' ou '1') des images chargées dans `data['data']`

Les données seront organisées comme suit :
Posons n=3707, la taille de notre dataset. Pour tout indice i compris entre 0 et n-1, l'image chargée dans `data['data'][i]` est de la classe `data['label'][i]`.

• Dans le cas du dataset 'Face Mask Detection' nous allons premièrement construire la liste `data['label']` associée aux classes des images de la base. Nous allons pour cela nous appuyer sur les fichiers `xml` du sous-répertoire `annotations` du dataset.
Ainsi, afin de parcourir ce dernier, nous faisons appel à la bibliothèque python `os` et en particulier à sa méthode `os.listdir(src)` renvoyant un itérateur sur les fichiers du répertoire.
Nous allons ensuite charger chacun de ces fichiers dans un dataframe dans le but d'en extraire les informations, en particulier celles liées aux masques détectés sur la photo :

```
6    None                None    NaN    NaN    NaN          NaN  without_mask
```

Etant donné que dans notre cas, seul le port ou le non port d'au moins masque sur les images importe, nous allons isoler les lignes du dataframes associées à un masque qu'il soit bien ou mal porté, puis les compter afin de déterminer le nombre de masques portés sur la photo :

```
nb_mask = series_mask.shape[0] # On obtient le nombre de masques portés dans l'image
```

Nous ajoutons alors à la fin de la liste `data['label']` la classe correspondante au fichier analysé : '1' si au moins un masque est présent, '0' sinon :

```
if( nb_mask>1 ):
    nb_mask=1
data['label'].append(nb_mask)
```

Nous construisons ainsi la liste des labels du dictionnaire `data`.

Nous chargeons maintenant l'ensembles des images du dataset dans la liste `data['data']`. Nous réutilisons pour cela la fonction `os.listdir(src)` afin de parcourir le sous-répertoires `images`.

```
# On verifie si le fichier actuellement lu est bien une image
if(picture[-3:] in ['png', 'jpg']):
    # On charge l'image courante en niveaux de gris dans 'img' un tableau numpy
    img = imread(current_path, as_gray=True);
```

des 3 canaux RGB et les charger en

Également dans le but d'améliorer l'entrainement du model, nous redimensionnons chaque image dans un format constant défini par les variables `width` et `height` :

```
data['data'].append(img) # On ajoute l'image à la liste des images chargées dans le dictionnaire
```

Nous construisons alors la liste `data['data']` des images du dictionnaire. Ces dernières ont été enregistrées dans un tableau numpy puis passées en niveaux de gris et redimensionnées.

• Le chargement en mémoire du dataset 'lfw' se fera de manière analogue à celui du dataset 'Face Mask Detection' à la différence que 'lfw' ne contient que des images de personnes non masquées.
La liste `data['label']` sera donc uniquement complétée avec des '0'.
De plus, nous ne souhaitons pas charger la totalité du dataset. Nous fixons alors une limite désignant le nombre maximum de sous répertoires parcourut.

Séparation du dataset en jeu d'entrainement et jeu de tests

Notre dataset ainsi construit et stocké en mémoire dans `data` doit maintenant être séparé en deux autres sous datasets. L'un servira à entrainer l'algorithme du model et l'autre à le tester.
Généralement cette subdivision du jeu de données n'est pas équitable et on laisse une part plus grande de données dans le jeu d'entrainement. C'est ce que nous effectuons ici en spécifiant la valeur du paramètre `test_size` à 0.2 (02% pour le jeu de test, 80% pour le jeu d'entrainement).
Par ailleurs, afin d'augmenter les chances d'entrainer l'algorithme du model sur tous les types de classe et d'image, on effectue, grâce au paramètre `shuffle=True,` un mélange aléatoire des couples donnée-classe avant de réaliser la subdivision.

Différences entre un jeu d'entrainement non mélangé et un jeu d'entrainement mélangé
Le jeu non mélangé peut potentiellement présenter plusieurs images de la même classe à la suite et d'autres concentrée en sa fin. La division du dataset ne permettrait alors pas d'entrainer le model sur toutes les classes
Source

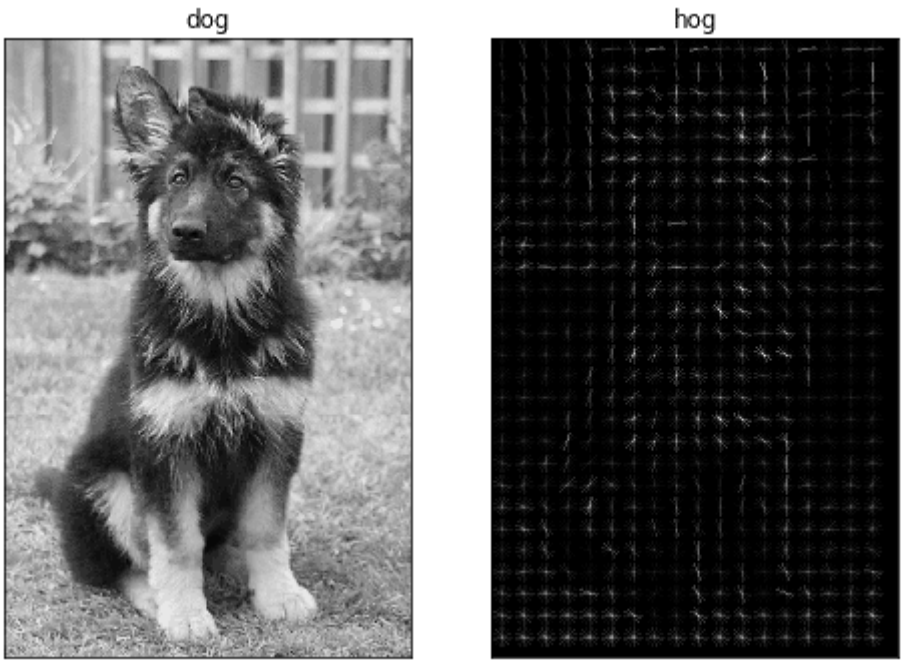
On obtient alors notre jeu d'entrainement `X_train` et notre jeu de test `X_test`.

Traitement préalable des images

Dans le domaine des algorithmes de reconnaissance d'objets et de vision par ordinateur une approche classique consiste à ne pas donner directement les images brutes au modèle pour l'entrainer, mais plutôt à effectuer des traitements préalables sur ces données afin d'en extraire des features. Ces features sont des structures de données construites à partir de l'image brute et en représentent une information spécifique sous une forme plus compréhensible par le modèle, comme par exemple ses contours, ses couleurs, …
L'intérêt de ces features est de décomplexifier l'analyse de l'image tous en maintenant un degré de précision suffisamment grand pour obtenir un modèle performant.
Parmis ces features, l'une des plus utilisées dans le cadre d'algorithmes de détection d'objet est le calcul d'un HOG de l'image.

Description des HOGs

Un HOG (pour Histogramme de gradients orientés) est obtenu en divisant l'image en plusieurs blocs de pixels dans lesquels le gradient de l'image sera calculé dans un nombre donné de directions.
Le résultat final décrit l'orientation des contours de l'image et peut être représenté graphiquement :



Source

Il existe deux types d'apprentissages automatisés :

A) Apprentissage supervisé

Un chercheur est là pour "guider" l'algorithme sur la voie de l'apprentissage en lui fournissant des exemples qu'il estime probants après les avoir préalablement étiquetés des résultats attendus. On dispose alors déjà d'un ensemble de données pour entraîner le modèle.

L'intelligence artificielle apprend alors de chaque exemple en ajustant ses paramètres (les poids des neurones) de façon à diminuer l'écart entre le résultat obtenus et le résultat attendus. La marge d'erreur se réduit ainsi au fil des entraînements, avec pour but, d'être capable de généraliser son apprentissage à de nouveaux cas.

L'apprentissage supervisé peut être divisé en deux sous-catégories : la régression et la classification :

• Si les labels sont discrets (des libellés ou valeurs finies) on parlera de **classification**. On peut utiliser différents types d'algorithmes, comme les plus proches voisins, les machines à vecteurs de supports, les arbres décisionnels, … la valeur de sortie est discrète. Ceci implique que le modèle est entraîné de telle manière à restituer une valeur de sortie discrète (y) pour une valeur d'entrée (x). => Il s'agit du système adopté dans notre projet de détection de masques.

• Si au contraire les labels sont continus (comme l'ensemble des nombres réels), on parlera de **régression**.

B) Apprentissage non supervisé

Contrairement à l'apprentissage supervisé, aucune donnée déjà classées/connues servant de base à la prédiction n'est connue. L'apprentissage par la machine se fait de façon totalement autonome. Des données sont alors communiquées à la machine sans lui fournir les exemples de résultats attendus en sortie.

La régression logistique

La régression logistique est une méthode d'analyse statistique qui consiste à prédire une valeur de données d'après les observations réelles d'un jeu de données. Elles sont aussi appelées modèle logit, sont un cas particulier des régressions linéaires.

Cela consiste à modéliser l'effet d'un vecteur de données/paramètres aléatoires sur une variable binomiale, c'est à dire pouvant avoir seulement 2 états, Vrai/Faux (0/1), d'où le terme logistique. 'y 'est la variable expliquée et les vecteurs (x1, x2, ..., xn) les variables prédictives/explicatives.

La régression logistique est un modèle mathématique qui combine un ensemble de variables prédictives (X) avec une variable aléatoire binomiale (Y). Elle est couramment utilisée dans le domaine de l'intelligence artificielle (IA) et du machine learning. Elle est considérée comme l'un des modèles d'analyse multivariée les plus simples à déchiffrer et analyser.

Quelques exemples :

- En médecine, elle permet par exemple de trouver les facteurs qui caractérisent un groupe de sujets malades par rapport à des sujets sains.
- Dans le domaine des assurances, elle permet de cibler une fraction de la clientèle qui sera sensible à une police d'assurance sur tel ou tel risque particulier.
- Dans le domaine bancaire, pour détecter les groupes à risque lors de la souscription d'un crédit.
- En économétrie, pour expliquer une variable discrète. Par exemple, les intentions de vote aux élections.

Différences entre régressions logistique et linéaire

La principale différence entre la régression logistique et la régression linéaire est que la régression logistique fournit un résultat constant, tandis que la régression linéaire fournit un résultat continu.
Dans la régression logistique, le résultat, tel qu'une variable dépendante, n'a qu'un nombre limité de valeurs possibles. Cependant, en régression linéaire, le résultat est continu, ce qui signifie qu'il peut avoir n'importe laquelle parmi un nombre infini de valeurs possibles.
La régression logistique est utilisée lorsque la variable réponse est catégorique, comme oui/non, vrai/faux et réussite/échec. La régression linéaire est utilisée lorsque la variable réponse est continue, comme le nombre d'heures, la taille et le poids.

HogTransformer(HOG) : Histogram of Oriented Gradients

Les HOG (histogramme de gradients orientés) sont des descripteurs de fonctionnalités qui rivalisent face aux réseaux de neurones profonds (DNN). Ils sont utiles pour établir des prévisions de données sur images floues et permettent de repérer les contours d'un élément d'une image et sont des alternatives plus simples mais surtout de l'ordre de 20 fois plus puissantes que les réseaux.

Pour cela, un descripteur HOG est calculé par des gradients d'image qui capturent les informations de contour et de silhouette d'images en niveaux de gris. Les informations de gradient sont regroupées dans un histogramme 1-D d'orientations, transformant ainsi une image 2D en un vecteur 1-D beaucoup plus petit. Ce vecteur forme l'entrée pour les algorithmes d'apprentissage automatique tels que les forêts aléatoires, les machines vectorielles de support ou les classificateurs de régression logistique.

descripteur HOG (à droite) (avec l'aimable autorisation de Digital Globe).

L'image est d'abord coupée puis redimensionnées.

1. On commence par calculer les gradients **vertical et horizontal** :

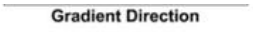
2. Ensuite, on trouve la **magnitude et la direction du gradient** : A partir de là, on peut dessiner les contours par changement d'intensité.

Gradient = Magnitude (longueur : définie l'importance du changement d'intensité) et direction (orientation : pointe vers le changement d'intensité, le moins intense. Cette orientation désigne un « gradient non signé » s'étendant de 0° à 180° (et pas 360°), puisque les mêmes angles se retrouvent avec des valeurs négatives) :

$$yx$$

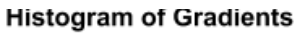
3. Calculer l'histogramme des gradients :

1 image est constituée de plusieurs cellules : Nombre de valeurs par pixel = nombre de cellules * 3 (si couleurs pour Rouge Vert Bleu) * 2 (magnitude et direction)



- A **gauche** : L'image
- Au **milieu** : Un zoom sur l'image afin de visualiser les pixels avec leurs gradients associés (magnitude et direction).
- A **droite** : Les gradients (magnitude et direction).

A l'aide de ces informations (les gradients, à droite), on va pouvoir mettre en place un tableau à 9 colonnes, c'est l'**histogramme des gradients**.



Explications :

1. Pour commencer, on cherche à insérer les valeurs des magnitudes correspondantes :
   - En bleu : la valeur à insérer vaut 2
   - En rouge : la valeur à insérer vaut 4

2. Ensuite, on recherche un emplacement dans l'histogramme afin d'insérer cette valeur.
   - Soit la direction correspond à une case bornée de l'histogramme et dans l'intervalle définit (ici de 0 à 160), au quel cas la valeur est directement insérée dans la case. → Par exemple en bleu, la direction vaut 80, alors la case correspondante dans le tableau est la 5ème.

   - Soit la direction ne correspond pas exactement à une case bornée de l'histogramme, au quel cas la valeur est divisée selon le pourcentage dans chaque case et insérée dans chacune des deux cases dans l'histogramme. → Par exemple en rouge, la direction vaut 10, alors elle empiète sur deux cases de l'histogramme. Il faut donc diviser la magnitude à 4 par 2 afin d'insérer la nouvelle valeur (2) dans les deux cases bornant la valeur 10. C'est pourquoi il faut entrer 2 dans les cases 0 et 20 de l'histogramme.

   - Soit la direction possède un angle supérieur à 160°, au quel cas la valeur insérée est aussi divisée selon le pourcentage dans chaque case. Cependant, comme on ne va pas au-delà de 180°, une partie va se trouver au niveau de la case 0. → Par exemple, en vert, la direction vaut 165, donc elle est supérieure à 160°. Elle empiète alors sur deux cases de l'histogramme. Il faut donc placer une partie dans 160 (à savoir 63,75), et l'autre dans 0 (donc 21,25).

Enfin, l'histogramme ressemblera à ceci :

3. On cherche maintenant à **normaliser** l'histogramme afin qu'il ne soit pas corrompu par les variations de luminosité de l'image. Pour cela, on examine des carrés de 16x16 pixels. Il y a donc 4 histogrammes à concaténer en un vecteur de 36 cases.
4. La fenêtre est ensuite décalée de 8 pixels afin de recommencer les opérations. A la fin, tous les vecteurs (de 36 cases) sont concaténés en un.

5. Ensuite, il suffit de le normaliser en un vecteur de 3 cases. Il s'agit de diviser chaque case de l'histogramme par la somme des carrés de chaque case. → Par exemple, le vecteur [128, 64, 32] possède une taille de $128^2+64^2+32^2=146,64$. Il faut alors le normaliser:

$$\left[\frac{128}{146,64}, \frac{64}{146,64}, \frac{32}{146,64}\right]$$

Calcul des HOGs : Les transformers

Dans le cas de notre projet nous souhaiterions alors calculer un HOG pour chacune des images des tableaux `X_train` et `X_test` afin de les fournier à notre model pour l'entrainer ou le tester.
Pour cela, sklearn implémente des objets permettant d'appliquer un traitement sur l'ensemble des éléments d'un tableau : les transformers. Ces derniers prennent en entrée un tableau de données, y appliquent un traitement puis retournent le tableau des données résultantes.
Sklearn propose plusieurs classes de transformers préalablement crées et permet notamment la définition de nouveaux transformers par héritage des classes BaseEstimator et TransformerMixin. Dans ce cas, la classe fille devra également définir les méthodes __init__, fit et transform.

Dans notre cas, nous définissons un transformer pour le calcul des HOGs des tableaux `X_train` et `X_test`.

```
            return np.array([local_hog(img) for img in X])
```

Puis nous instancions les transformers à utiliser. A noter que nous utiliserons le transformer StandardScaler, permettant de changer la taille des images d'un tableau :

```
scalify = StandardScaler()
```

Suite à cela, nous réalisons les traitements préalables sur les données du jeu d'entrainement `X_train`. Pour cela, nous appelons successivement la méthode `fit_transform` des transformers sur le jeu d'entrainement `X_train`, le transformant pas à pas :

```
X_train_prepared = scalify.fit_transform(X_train_hog)
```

Entrainement et test du modèle

Maintenant que nos données ont été préalablement traitées, nous pouvons entrainer notre model sur ces dernières.
Nous choisissions d'utiliser le modèle 'SGDClassifier' qui réalise une classification par descente de gradient stochastique.
Ce dernier est implementé par sklearn au travers de la classe `SGDCClassifier`.

Entrainement

L'entrainement de notre modèle se fait à l'aide de la méthode `fit` de la Classe `SGDCClassifier`.

```

```

Test

Nous réalisons par la suite des tests sur le jeu de test grâce à la méthode predict de la classe `SGDCClassifier`.

```
print("Pourcentage precision: ", 100*np.sum(y_pred == y_test)/len(y_test))
```

Test avec images externes

La dernière cellule de la fiche permet de faire des tests avec des images externes :

```
    print("masque nondetecte")
```