SY32 - Analyse et synthèse d'images - Détection de visages

Introduction

Le but de ce projet est de développer un modèle capable de détecter avec précision un ou plusieurs visages dans une image. Après avoir détaillé mes choix d'implémentation, je présenterai ma démarche d'entraînement du modèle pour enfin présenter mes résultats et conclusions.

Choix d'implémentation

Résumé

J'ai choisi d'utiliser un réseau de neurones à convolution en utilisant l'architecture *VGG16* car elle présente de bon résultats sur le dataset *cifar-10* qui est un dataset d'images en 32x32.

Mon idée est qu'au lieu de prendre les boîtes labels comme exemple positifs et générer des exemples négatifs aléatoires, je vais générer tous les exemples possiblements vus par une *sliding window* ayant certains paramètres.

Ainsi le modèle est directement entrainé a voird es exemples venant d'une sliding window.

Il m'a donc d'abord fallu déterminer les paramètres de *sliding window* optimaux afin de réaliser mon découpage d'images.

J'ai également fait le choix de faire varier la taille de la fenêtre car je vais de toute façon tout redimensionner en 32x32 qui sera la taille d'entrée de mon modèle.

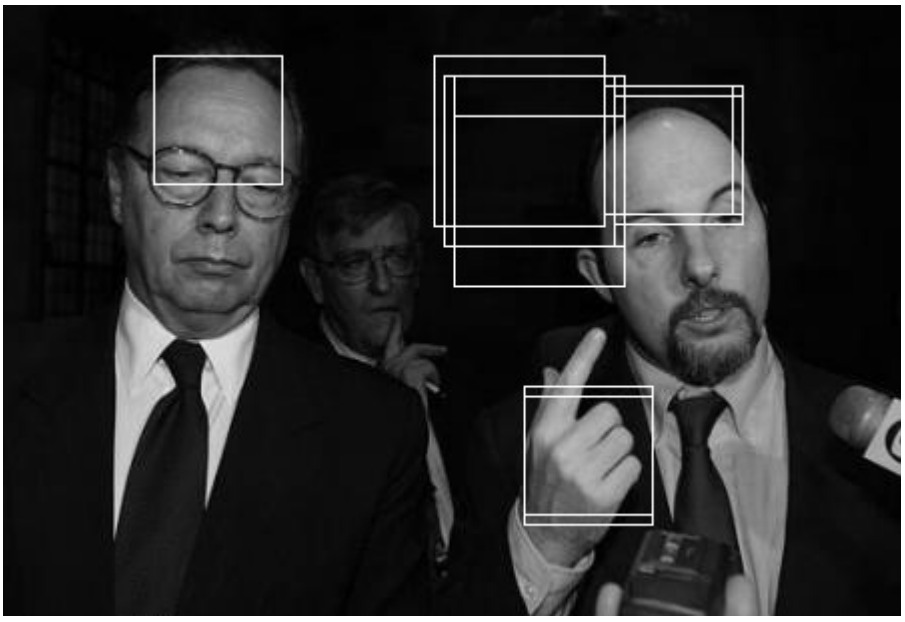
Par la suite je vais entraîner le CNN sur les exemples précedemments générés en mesurant le f_1 qui servira de mesures proxy pour mes prédictions. En effet j'assume le fait que maximiser le f_1 lors de mon entraînement le maximisera lors de mes détections.

Enfin grâce au seuil maximisant le f_1 pour un modèle donné je peux appliquer mon algorithme de *sliding window* à mon ensemble de validation pour mesurer sa performances et modifiers quelques hyper paramètres avant de faire un prédiction sur l'ensemble de test.

Choix du modèle

J'ai dans un premier temps opté pour l'implémentation rapide d'un modèle de boosting en utilisant la librairie LightGBM. L'avantage de ce type de modèle et de cette librairie et qu'il est très facile et rapide de développer et entrainer un modèle. Cette démarche m'a permis d'avoir une baseline et de cerner le problème et pour en définir la complexité.

Pour l'entraînement du modèle de boosting j'ai simplement extrait les visages des images en utilisant les coordonnées des boîtes labels puis j'ai redimmensionné l'image dans un format 64x64. Les prédictions de mon modèle en utilisant de la cross validation étaient concluantes mais en pratique les résultats n'étaients pas bons :



Détection décalée ou détection de mains / doigts.

Constatant les faibles performances de ce modèle deux choix s'offrent à moi : Faire un second entraînement avec ce modèle ou changer de modèle.

J'ai choisi de changer de modèle et de me diriger vers un réseau de neurones à convolution. En effet ces dernières années, les réseaux de neurones à convolution ce sont imposés comme le meilleur moyen de détection et de classification d'image. Le développement du premier modèle de boosting a permis d'avoir une base pour

comparer mes performances et de commencer à développer les algorithmes comme la *sliding window* que j'utiliserai par la suite.

Plan d'action

- Analyser les données
- Déterminer le pas et la taille de fenêtre optimale
- Créer le Dataset
- Entraîner un classifieur
- Analyser les performances du classifieur
- Appliquer l'algorithme de *sliding window* sur les images
- Mesurer les performances de détection

Analyse des données

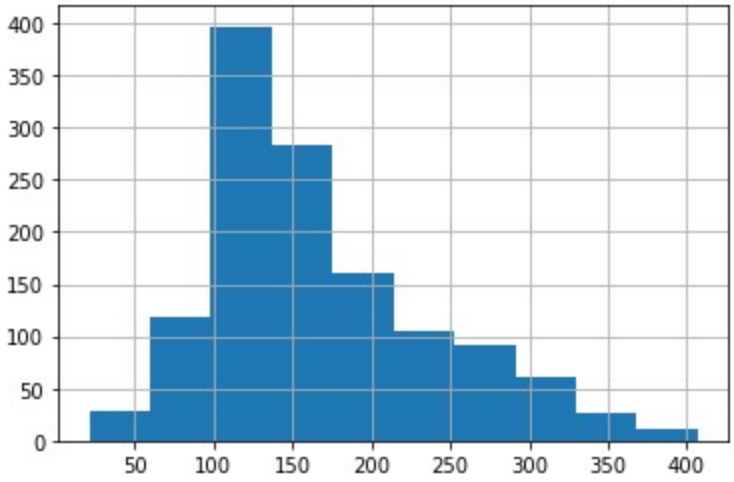
Pour bien démarrer l'important est d'analyser et d'explorer nos données d'entraînement. On va notamment s'intéresser aux labels que l'ont doit déterminer :

	k	i	j	h	l
count	1284.000000	1284.000000	1284.000000	1284.000000	1284.000000
mean	494.914330	47.172897	130.934579	167.451713	110.725857
std	290.226415	51.265008	78.676387	72.207741	47.852849
min	1.000000	0.000000	0.000000	21.000000	13.000000
25%	243.750000	18.000000	72.000000	115.000000	77.000000
50%	488.500000	30.000000	112.000000	147.000000	97.000000
75%	746.250000	54.000000	176.000000	207.000000	137.000000
max	1000.000000	334.000000	418.000000	407.000000	275.000000

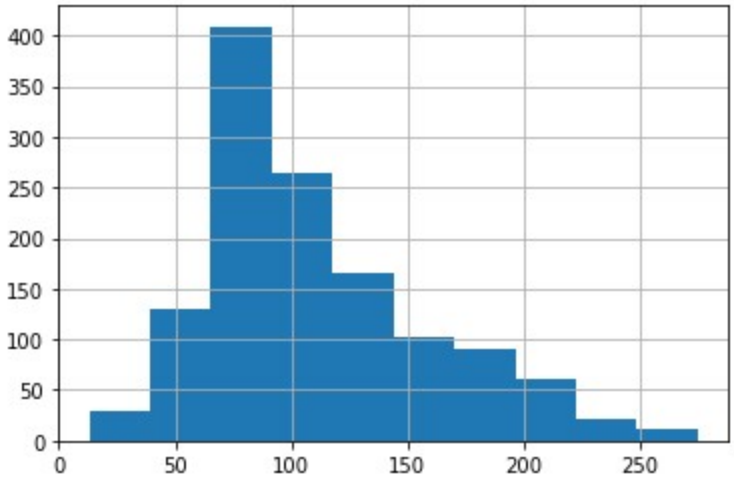
On peut déduire qu'en moyenne, la boîte label qui englobe le visage fait *167* pixels de hauteur pour *110* pixels de large.

On peut aussi utiliser des graphiques pour analyser la distribution des boîtes labels.

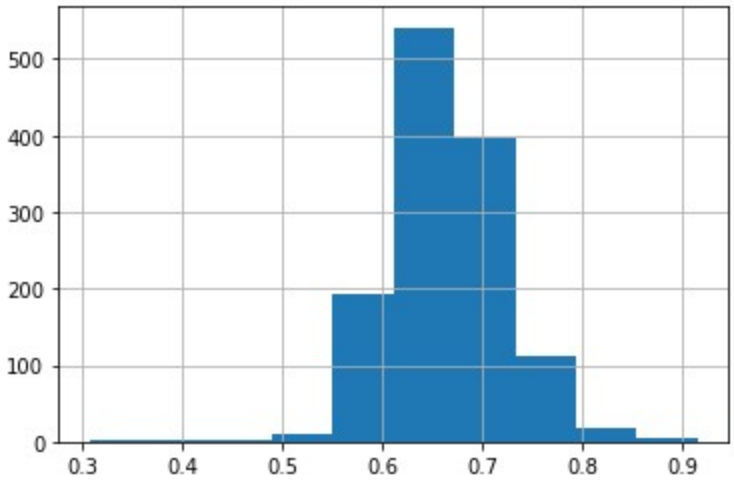
Histogramme de la largeur des boîtes:



Histogramme de la hauteur des boîtes:



Histogramme du ratio largeur/hauteur des boîtes:



On peut voir que le ratio largeur/hauteur est majoritairement compris entre *0.55* et *0.75*. On va donc essayer de conserver ce ratio lors de la définition de la taille de nos fenêtres. Ce ratio peut s'expliquer par la forme d'un visage qui est en général plus long que large. Les boîtes sont donc en général des recangles.

Déterminer les paramètres optimaux de l'algorithme *sliding window* et création du dataset

J'ai choisi d'utiliser une autre manière d'entraîner mon modèle que celle suggérée en cours. En effet en cours on nous suggère de prendre les boîtes labels comme exemple positif et de générer des exemples négatifs aléatoires. Puis ensuite de refaire un entraînement en incluant les faux positifs dans notres dataset d'entraînement. Cette manière de faire présente quelques inconvénients : Comment choisir les fenêtres négatives ? Combien de fenêtres négatives faut-il générer ? Quelle proportion positif / negatif faut-il ?

Il n'est pas facile de répondre à ces questions, c'est pour cela que j'ai décidé d'utiliser une autre méthode : Puisque nous allons utiliser un algorithme de *sliding window* sur nos images, nous allons générer tous les découpages possiblements vus par notre algorithme et entraîner notre modèle dessus. Ainsi notre modèle aura conscience de la proportion visage / non visage que l'on aura en production et sera entraîné à reconnaître des visages alors que la fenêtre n'est pas parfaitement centrée sur celui-ci.

Algorithme *sliding window*

Avant de se lancer dans l'entraînement d'un modèle il vaut mieux d'abord s'attarder sur la manière avec laquelle nous allons effectuer nos prédictions.

Nous allons donc utiliser un algorithme de *sliding window* afin d'extraire des images avec un certain pas et certaines tailles de fenêtre.

Il y a 2 manières d'implémenter cet algorithme :

1. Choisir une taille de fenêtre fixe et faire varier l'échelle de l'image pour simuler des tailles de fenêtre différentes.
2. Fixer l'échelle de l'image et faire varier la taille de la fenêtre qui va la parcourir.

L'implémentation d'un classifieur implique que l'image d'entrée doit toujours avoir le même format ce qui naturellement nous dirige vers la méthode 1.

Or avec les outils de redimensionnement d'image de la librairie Pillow on peut aisément appliquer la méthode 2. En effet il suffit de redimensionner l'image capturée par la fenêtre avant de faire la prédiction. Utiliser cette méthode a un avantage qui est qu'au lieu de manipuler et faire varier des échelles on va faire varier une taille de fenêtre ce qui est un peu plus interprétable qu'une échelle.

De plus grâce à notre étude de la répartition des boîtes on peut facilement définir des taille de fenêtre qui couvrent le plus large spectre possible de hauteurs et largeurs.

Déterminer les paramètres optimaux

Afin de déterminer les paramètres optimaux de *pas* et *taille* de la fenêtre on va simuler tous les découpages possiblement générés par un jeu de paramètres puis mesurer leur performance.

En effet on suppose que l'on détient déjà un modèle parfait qui détecte 100% des visages. En fixant une ou plusieurs tailles de fenêtre et un ou plusieurs *stride* (pas de déplacement de la fenêtre) on peut simuler toutes les fenêtres (ou boîtes) qui vont être générées par ce découpage.

Pour déterminer la qualité de notre découpage on peut ensuite mesurer dans celui-ci, combien de nos boîtes labels sont suffisament recouvertes par nos fenêtres générées.

On peut ainsi déterminer un découpage optimal ayant le meilleur compromis entre précision maximale et nombre de fenêtres générées.

On peut évidemment avoir une précision maximale de 100% en générant plus de 300 000 images mais le temps pour la prédiction serait extrememement long.

En utilisant la technologie du multiprocessing on peut tester un grand nombre de paramètres différents en un temps réduit et ainsi déterminer la combinaison optimale de taille de fenêtre et stride.

Exemple de résultat d'un jeu de paramètres
ws pour 'window size'
acc pour 'accuracy' représente la précision maximale obtenue par cette taille de fenêtre
mean pour 'moyenne' nombre moyen de boîtes trouvées par visage

	stride	ws	acc	mean	num_im
3	30	[(150, 97)]	0.460526	1.438694	7145
1	30	[(200, 130)]	0.323835	1.251451	3708
2	30	[(150, 150)]	0.269549	1.429010	3857
7	30	[(100, 100)]	0.264887	1.295771	11873
19	30	[(100, 65)]	0.260752	1.347174	20256
4	30	[(250, 162)]	0.181579	1.452174	1853
6	30	[(300, 195)]	0.128271	1.390973	1039
0	30	[(200, 200)]	0.125414	1.475420	2004
5	30	[(250, 250)]	0.073158	1.340185	892
11	30	[(350, 227)]	0.066316	1.162132	512
20	30	[(50, 50)]	0.025414	1.144970	54998
9	30	[(300, 300)]	0.022782	1.382838	444
21	30	[(50, 32)]	0.019248	1.214844	95795
13	30	[(400, 260)]	0.018496	1.288618	341
8	30	[(350, 350)]	0.003158	2.428571	150
12	30	[(400, 400)]	0.002707	1.416667	79
10	30	[(450, 450)]	0.000000	NaN	0
14	30	[(500, 500)]	0.000000	NaN	0
15	30	[(450, 292)]	0.000000	NaN	79
16	30	[(550, 550)]	0.000000	NaN	0
17	30	[(500, 325)]	0.000000	NaN	99
18	30	[(550, 357)]	0.000000	NaN	55

La combinaison retenue est la suivant :

- Taille de fenêtres :
  - (100, 85)
  - (200, 100)
  - (250, 212)
- Stride : 35

Le stride n’est pas définit en nombre de pixel mais en pourcentage de la taille de la fenêtre. C’est à dire qu’a chaque itération la fenêtre va se déplacer de 35% de sa largeur ou hauteur. Cela permet de réduire le nombre d’images générées par des fenêtres qui se superposeraient trop et d’harmoniser le déplacement de la fenêtre qui s’adapte à la taille de la fenêtre.

La taille des fenêtres retenues est varié et couvre la majeure partie du spectre des tailles défini dans l'analyse des données. En guardant ces tailles d'images et en faisant varier le stride on obtient les résultats suivants :

Stride	Précision max	Nombre moyen de fenêtre par visage	nombre d'image total
30	0.936	2.023	201 673
35	0.900	1.565	153 061
40	0.807	1.311	122 755

Le découpage avec un stride de 35 semble le plus raisonnable niveau précision max / nombre d'image. De plus on pourra facilement faire varier le stride plus tard lors de la validation de notre modèle.

Création du dataset

Maintenant que l'on a défini nos paramètres optimaux, on peut créer notre dataset. Pour cela on va appliquer le découpage que l'on a déterminé précédemment.

Pour chaque image du dataset, On calcule les coordonnées de toutes les boîtes qui pourraient être vues par notre algoritme en fonction de la taille d'image et le stride défini. Une fois ces coordonnées déterminées, on calcule les IoU (Intersection over Union) entre nos boîtes générées et nos boîtes labels afin de déterminer quelles boîtes sont positives ou négatives. Ainsi les boîtes générées ayant une aire de recouvrement suffisante avec une boîte label auront la classe *1* et seront considérées comme un visage; les autres boîtes auront la classe *0* donc non visage.

Le seuil d'IoU attendu pour être considéré comme un vrai positif lors de la prédiction finale est de 0.5. Cependant, pour éviter les confusions, j'ai décidé pour l'entrainement du modèle qu'un recouvrement de 0.4 était suffisant pour qu'une boîte soit considéré comme contenant un visage.

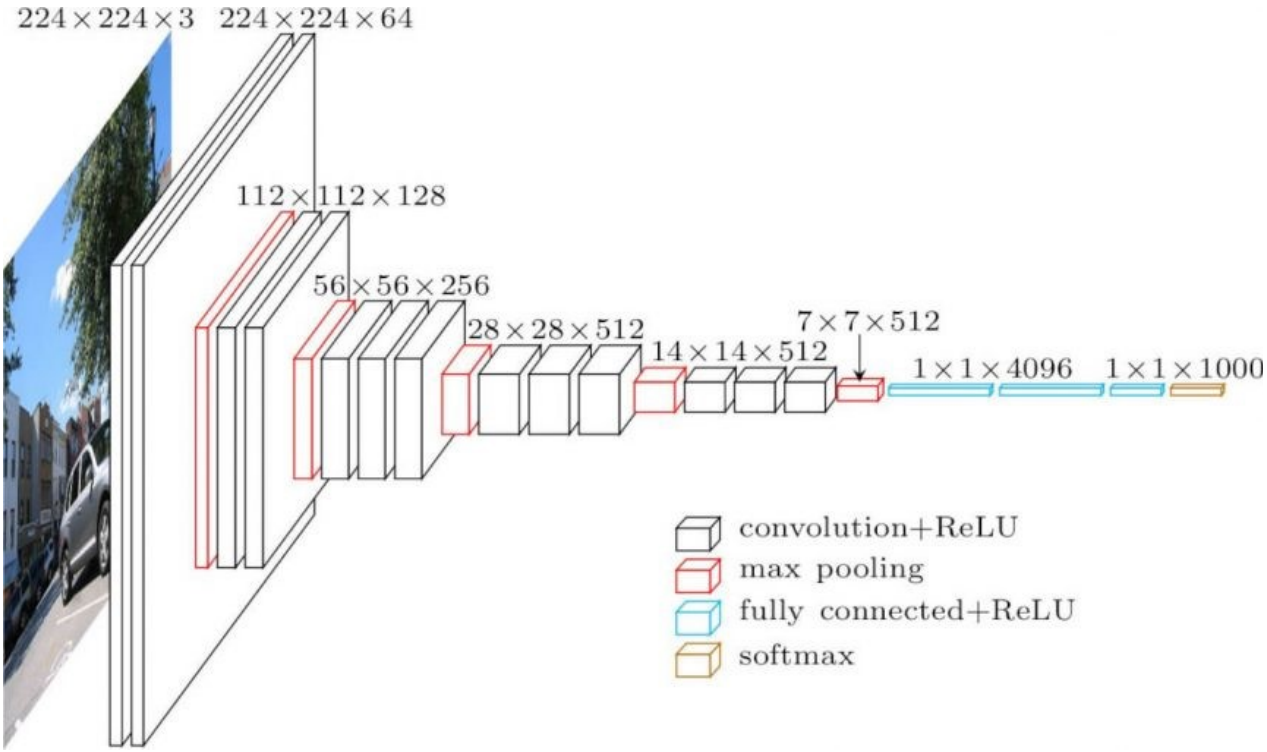
De plus cela permet au modèle d'avoir plus de données ayant la classe visage. Avoir une tolérance de 0.4 d'IoU permet d'être plus permisif au niveau des détections, on appliquera plus tard un algoritme pour fusionner les multiples boîtes détectées, ce qui nous permet de conserver des boîtes un peu décalées ou un peu trop petite ou trop grande.

On peut enfin séparer notre dataset en 2 jeux de données : Un jeu de donnée d'entrainement et un de validation. Il faut veiller à ce que les images découpées

correspondant à la même image soient ensemble dans le même set sinon ce serait tricher et le modèle pourrait reconnaître les images. En faisant cela on conserve aussi la proportion d'image contenant des visages dans chaque dataset. Ainsi les images 1 à 800 appartiennent au dataset d'entraînement et les images 801 à 1000 au dataset de validation.

Architecture du CNN

Pour l'architecture du CNN j'ai choisi d'utiliser une architecture VGG16 telle que décrite ici [1] (https://arxiv.org/abs/1409.1556)



L'architecture VGG16 étant de base conçue pour de grandes images *(224, 224, 3)* j'ai tout de même choisi d'utiliser cette architecture car elle présente de bons résultats [2] (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7486599&tag=1) sur le dataset CIFAR-10 qui est un dataset d'images en taille *(32, 32, 3)* avec 10 classes différentes. Cette architecture paraît donc appropriée à notre cas d'utilisation et c'est celle que je vais implémenter. Pour adapter l'architecture à une classification binaire il suffit de modifier la dernière couche qui est un *softmax* en *sigmoid* afin d'avoir une seule sortie correspondant à la probabilité que l'image soit un visage

Entraînement du modèle

Pour entraîner le modèle j'ai utilisé les `ImageDataGenerator` de l'API Keras qui

permettent de modifier aléatoirement les images à chaque batch. Ainsi, on peut par exemple inverser horizontalement l'image ou appliquer une légère rotation. Cela ne dénature pas l'image, un visage reste un visage, mais pour notre modèle c'est une image complétement nouvelle qu'il n'a jamais vu. Cela permet de prévenir l'overfitting aux données d'apprentissage, c'est à dire que le modèle apprend à reconnaître les images d'entraînement et ne parviens pas à généraliser à des images qu'il n'a jamais vu.

Il est important d'effectuer ces transformations à la volée afin de ne pas perturber la distribution des classes d'images. En effet on pourrait artificiellement augmenter le nombre d'images positives afin d'avoir un dataset plus équilibré mais notre modèle s'attendrait alors une fois en production à voir plus de visages que ce qui est vraiment présent et ce n'est pas ce que l'ont veut.

Attention cependant, lors du test en validation il ne faut pas appliquer ces transformations aléatoires sinon on ne peut pas comparer nos performances entre elles. En effet notre modèle peut être moins performant car nos transformations aléatoires ont généré un dataset plus compliqué. Il est donc important de toujours utiliser le même dataset de validation.

Gestion du déséquilibre des données

L'une des problématique d'avoir un jeu de données tant déséquilibré est que notre modèle ne voit pas suffisament de visages et qu'en prédisant toujours la classe *0* il aura quasi 95% de précision.

Pour contrer ce déséquilibre on va demander à notre modèle de donner plus d'attention aux exemples positifs que négatifs. Dans ce but nous allons défénir des poids que nous allons appliquer à chacune des classes grâce à la formule suivante :

$$w_0 = (1 - \alpha) \cdot (n_1 + n_0)/n_0$$
$$w_1 = \alpha \cdot (n_0 + n_1)/n_1$$

avec α un coefficient représentant l'importance que l'on souhaite donner à la classe *1* et que l'on peut faire usuellement varier entre 0.2 et 0.5. Avec un α de 0.5 on a la somme des poids de la classe *1* qui est égale à la somme des poids de la classe *0* et la somme des poids totaux qui reste égale à la somme des poids normaux. On peut résumer les conditions par ce système d'équation pour:

$$\alpha \cdot n_1 \cdot w_1 = (1 - \alpha) \cdot n_0 \cdot w_0$$
$$\alpha \cdot n_1 \cdot w_1 + (1 - \alpha) \cdot n_0 \cdot w_0 = n_0 + n_1$$

On remarque que lorsque $\alpha = 0.5$ on a la classe *1* qui aura la même importance que la classe *0* même si elle est sous représenté car elle possède la moitié des poids totaux.

Une fois ces poids calculés on va faire apprendre notre modèle et l'optimiser en utilisant la fonction d'erreur *Weighted Binary Cross Entropy* :

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) \cdot w_1 + (1 - y_i) \cdot \log(1 - p(y_i)) \cdot w_0$$

Avec y_i le label *0* ou *1* et $p(y_1)$ la probabilité calculée par le modèle.

Mesures des performances

Pour mesurer les performances de notre modèle on a vu qu'il ne servait à rien de regarder la précision seule car un modèle prédisant toujours la classe *0* aura presque 100% de précision au vu du rapport visage/non visage de notre dataset.

On va donc plutôt calculer à chaque époque la valeur du score f_1 optimal obtenue par notre modèle sur l'ensemble de validation. Pour calculer le score f_1, on doit pouvoir faire la distinction entre vrais positifs, faux positifs et faux négatifs. Nous devons donc déterminer un seuil à partir duquel une prédiction est considérée comme positive, ce seuil permettant d'obtenir le meilleur score $f1$ pour ce modèle. Pour cela on trie nos prédictions par score et on calcule la précision et le rappel en utilisant chaque prédiction comme nouveau seuil:

$$\text{precision} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux positifs}}$$
$$\text{Rappel} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux négatifs}}$$

On calcule ensuite le score f_1 pour chaque seuil et on trouve le plus performant :

$$f_1 = 2 \cdot \frac{\text{precision} \cdot \text{rappel}}{\text{precision} + \text{rappel}}$$

On peut également mesurer le f_2 avec $\beta = 2$:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{rappel}}{(\beta^2 \cdot \text{precision}) + \text{rappel}}$$

Le f_2 permet de favoriser le rappel en sacrifiant un peu de précision et ainsi reconnaître plus de visages au risque de se tromper plus souvent. Cette métrique est intéressante mais le seuil trouvé avec celle-ci sera moins performant au final que celui trouvé avec la métrique f_1

Exemple de résultat

```
y_true       1.000000
y_pred       0.893446
vp        1130.000000
fp         208.000000
fn         103.000000
prec         0.844544
rec          0.916464
f1           0.879035
Name: 29761, dtype: float64
```

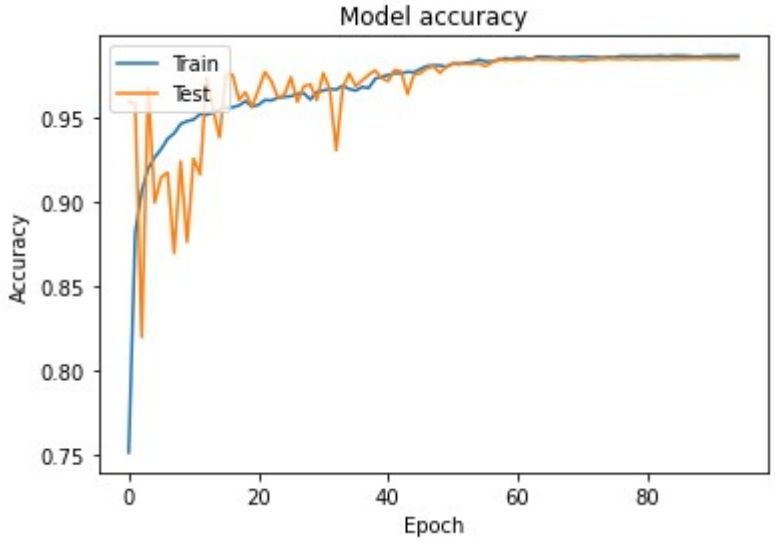
On voit que l'on obtient un $f_1 = 0.879$ pour un seuil de prédiction à 0.893 (ligne *y_pred*). Avec ce seuil on voit que l'on a une précision de 0.845 et un rappel de 0.916 ce qui signifie que l'on trouve 91% des visages et que si le modèle prédit un visage il a raison dans 84% des cas.

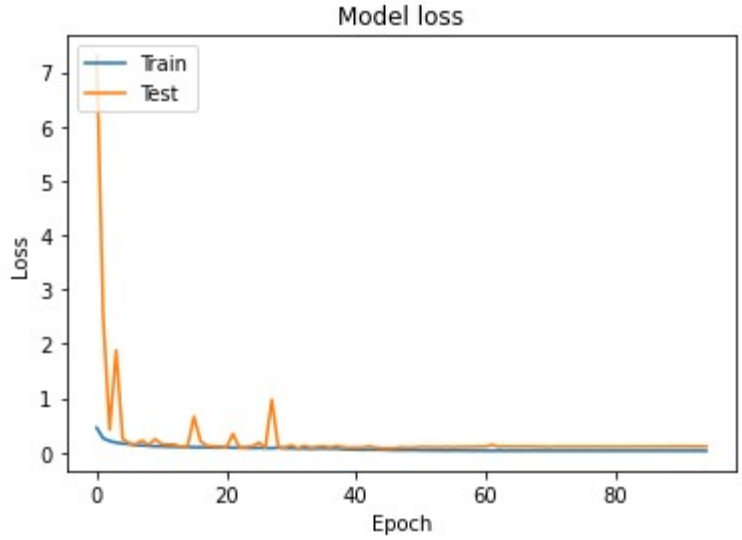
On peut s'intéresser au f_2, on voit que le score est plus élevé et que le seuil de prédiction est plus bas 0.711 mais on peut aussi voir que la précision est déscendue à 0.784 ce qui est trop bas pour notre utilisation. En effet on ne veut pas que notre modèle signale trop de visage là où il n'y en a pas.

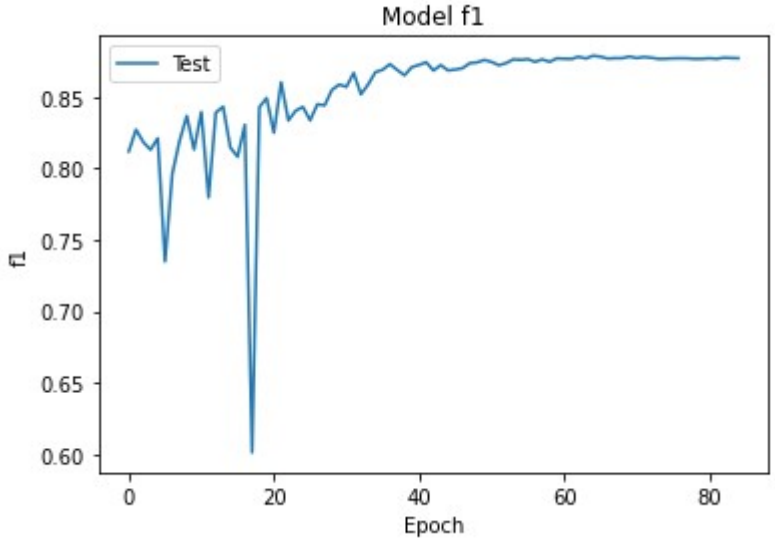
```
y_true       1.000000
y_pred       0.711501
vp        1172.000000
fp         322.000000
fn          61.000000
prec         0.784471
rec          0.950527
f2           0.911920
Name: 29888, dtype: float64
```

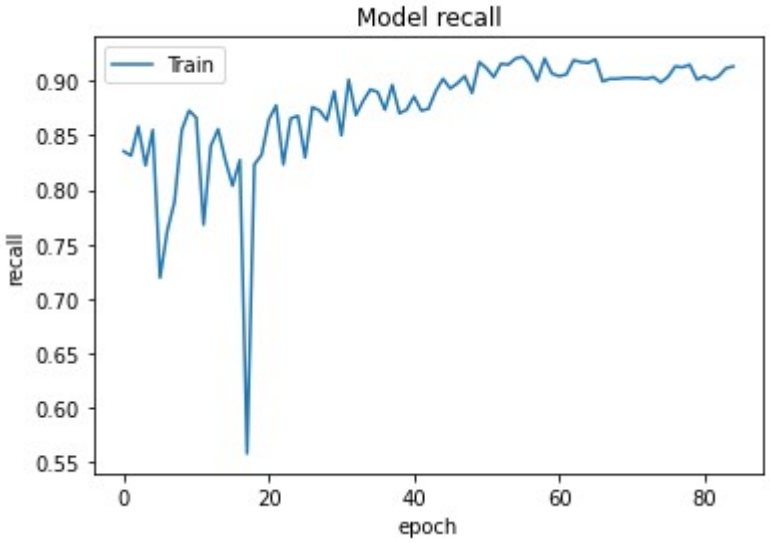
Résultats

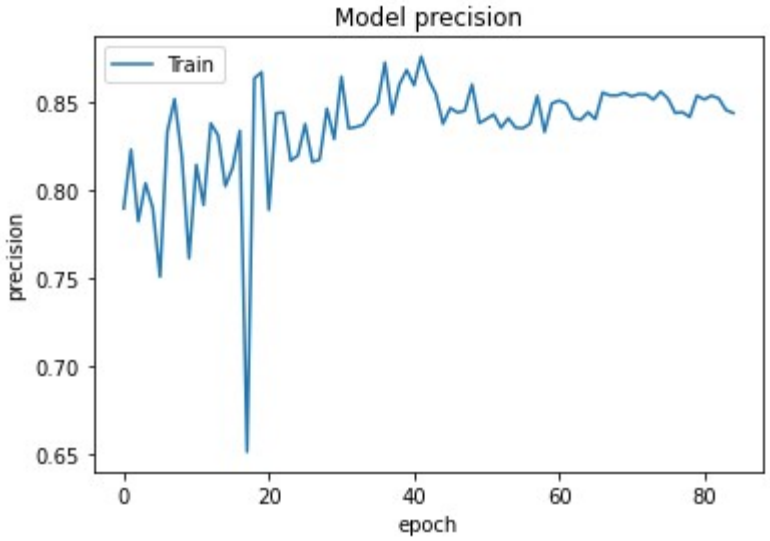
Après avoir entraîné le modèle pendant 100 époques pour une durée totale d'environ 4 heures on obtient les courbes suivantes :


Model accuracy
Train
Test
Accuracy
Epoch
0.75
0.80
0.85
0.90
0.95
0
20
40
60
80


Model loss
Train
Test
Loss
Epoch
0
1
2
3
4
5
6
7
0
20
40
60
80


Model f1
Test
f1
Epoch
0.60
0.65
0.70
0.75
0.80
0.85
0
20
40
60
80





Le modèle qui a montré les meilleurs résultats est celui de l'époque 47 avec un bon compromis f_1 de 0.873 et *validation loss* de 0.0766.

Le modèle de l'époque 74 avait un meilleur f_1 de 0.879 mais la *validation loss* a légèrement remonté (0.1061) ce qui indique un léger *overfitting*.

Prédiction

Maintenant que notre modèle est entraîné et peut reconnaître des visages sur des imagess en 32x32 nous pouvons appliquer notre algorithme de *sliding window* cette fois pour faire des prédictions.

Calibrage de la détection

Paramètres utilisés

Pour déterminer les paramètres de détection nous allons utiliser les paramètres que nous avons utilisé pour le découpage qui a servi à créer le dataset car c'est avec cela que notre modèle s'est entrâiné :

- Taille de fenêtres :
  - (100, 85)
  - (200, 100)
  - (250, 212)
- Stride : 35

Seuil de détection

Pour déterminer le seuil de détection optimal il faudrait prédire toutes les boîtes possible puis les trier par score et calculer le f_1 pour chaque score unique afin de trouver quel seuil maximise sa valeur.

Faire cette opération est très gourmand en ressources et nécessite beaucoup de temps. C'est pourquoi nous allons utiliser le seuil trouvé précédemment qui maximise la détéction sur l'ensemble de validation lors de l'entraînement du modèle. On utilise le score f_1 de notre modèle en validation comme un proxy et on assume que le seuil maximisant le f_1 en validation, maximisera le f_1 lors de la prédiction.

Sélection des boîtes

Notre algorithme va calculer un ensemble de boîtes dans lesquelles il estime voir un visage. Nous allons inévitablement nous retrouver avec plusieurs boîtes par visage. Une idée serait de garder uniquement la boîte ayant le meilleur score pour chaque visage. Cependant, lors de l'entraînement, il a été décidé que les boîtes ayant un recouvrement IOU supérieur à 0.4 avec une boîte label seraient considérées comme positives. Notre modèle va donc avoir tendance à détecter des boîtes légèrement décalées.

On va donc appliquer une technique de fusion des boîtes, en effet notre modèle détecte des boîtes de différentes taille autour du visage avec une forte concentration vers le centre du visage. Faire la moyenne des coordonnées de toute ces boîtes est la méthode qui a montré les meilleurs résultats sur l'ensemble de validation et celle que l'on va garder.

De plus pour savoir quelles boîtes fusionner entre elle, on commence par uniquement regarder les boîtes les plus petites qui ne se recouvrent pas, puis on va les fusionner avec les boîtes de plus en plus grande jusqu'à en dégager des groupes

de boîtes qui correspondent chacun à un visage.

Enfin une fois les boîtes fusionnées, on refait une prédiction sur les images en les découpant avec les coordonnées de ces boîtes afin de vérifier que notre score de prédiction est toujours au dessus du seuil précédemment défini, sinon on supprimme la boîte.

Calcul du score f_1

Il nous faut pouvoir calculer le score f_1 des prédictions de notre modèle. On rappelle qu'une boîte est considérée comme vrai positif si elle recouvre une boîte label avec un IOU d'au moins 0.5. Il ne doit y avoir qu'une seule détection par boîte label et si un visage est detecté plusieurs fois, les détections supplémentaires seront considérées comme faux positifs. Enfin un visage non détecté sera considéré comme faux négatif.

Pour avoir une idée de nos performances nous allons faire des prédictions sur l'ensemble de validation afin de pouvoir estimer notre score f_1 potentiel sur l'ensemble de test.

Après avoir essayé différents paramètres de taille de fenêtre, de stride et de fusion de boîtes prédite, il s'avère que les paramètres donnant les meilleurs résultats sont les suivants :

- Taille de fenêtres :
  - (100, 85)
  - (200, 100)
  - (250, 212)
- Stride : 30
- Méthode de fusion des boîtes : moyenne des coordonnées

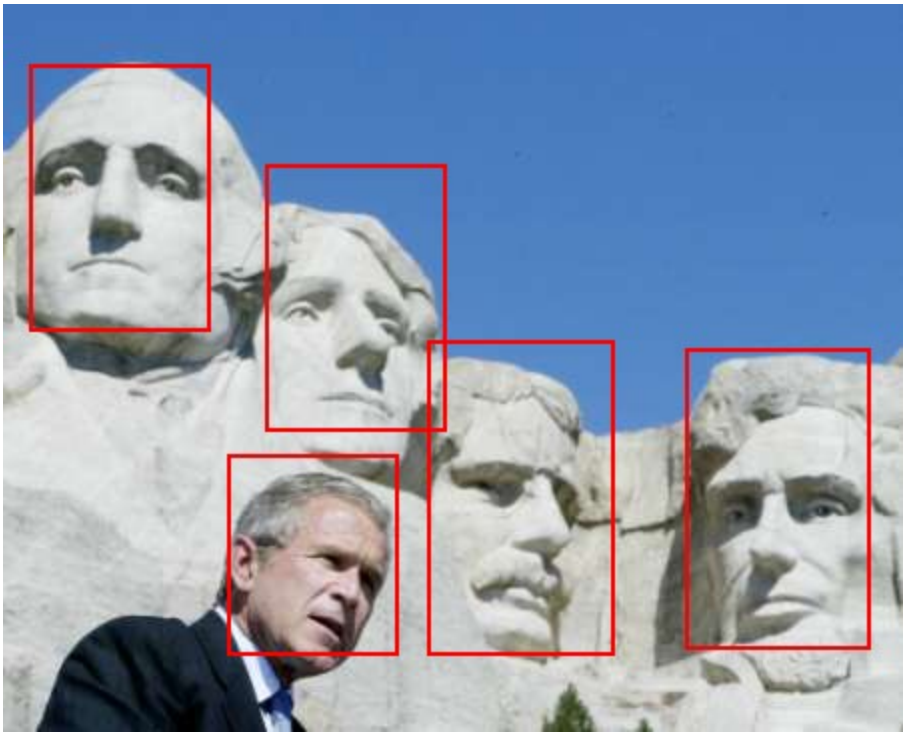
On voit que le stride a été diminué à 30 ce qui permet d'avoir un pas plus petit. En effet lors de l'entraînement on a choisi un pas de 35 pour ne pas généré trop d'images d'apprentissages mais pour la détection un pas de 30 n'augmente pas significativement de temps de prédiction et apporte un boost au score f_1.

Performances de détection

Le score f_1 trouvé en utilisant les paramètres optimaux précédemment défini sur l'ensemble de validation est un bon estimateur du score f_1 final que l'on obtiendra sur l'ensemble de test.

On a obtenu sur l'ensemble de validation un score autour de 0.94 ce qui a donné sur l'ensemble de test un score exact de 0.9561.

Le temps pour réaliser la détection est relativement rapide avec environ 3 minutes pour faire une détection sur les 500 images de test.



Conclusions

L'objectif de ce projet a été atteint, il s'agissait de développer un détecteur de visages sur des images de différentes tailles. Il a fallu être astucieux dans la création des jeux de données d'entraînement et validation en utilisant seulement les 1000 images à notre disposition afin d'en tirer le meilleur parti. Au final on a obtenu un modèle relativement robuste, qui peut détecter un visage plus ou moins gros et ne détecte pas de visages par exemple dans le paysage ou dans du texte. Une des faiblesse actuelle du modèle est qu'il ne reconnaît pas les visages plus petits, il faudrait donc ajouter une taille de fenêtre plus petite mais cela implique beaucoup plus de temps d'entraînement et de temps de prédiction.

Il reste encore beaucoup de possibilités pour améliorer notre détecteur, on pourrait chercher à calculer les paramètres donnant les meilleurs résultats durant l'application de la *sliding window*. On pourrait tenter d'utiliser d'autres paramètres de génération du dataset pour apprendre à notre modèle à être plus précis ou mieux centré sur les visages.

On pourrait également explorer d'autres approches comme par exemple prendre en entrée l'image entière et directement prédire les coordonnées des boîtes.

Instructions

python 3.7.5
tensorflow 2.0.0
tensorflow-gpu 2.0.0
numpy 1.18.1
pandas 1.0.3
pillow 7.1.1
keras-gpu 2.3.1

Fichier	**fonction**
create_dataset	Crée le découpage et sauvegarde le dataset dans un dossier train32x32 (multiprocessing)
find_params	Calcule les performances maximale de certains paramètres de découpage (multiprocessing)
callback	Custom Keras Callback pour calcul du f_1 à la fin de chaque époque
model	Modèle Keras utilisé pour l'entraînement et la prédiction
util	Ensemble de fonctions utilitaires
visage_detector	Classe qui effectue la détection
train	Lance l'entraînement
test	Lance la détection et génère le fichier sub.txt

Pour créer le dataset d'entraînement il faut lancer le fichier *create_dataset.py*. On peut ensuite lancer l'entraînement avec le fichier python *train*

L'entraînement peut prendre beaucoup de temps selon la configuration.
~4h avec GTX 1060 6Go

Utiliser le fichier python *test* pour faire la prédiction

Références

[1] (https://arxiv.org/abs/1409.1556) Very Deep Convolutional Networks for Large-Scale Image Recognition by Karen Simonyan, Andrew Zisserman

[2] (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7486599&tag=1) Very Deep Convolutional Neural Network Based Image Classification UsingSmall Training Sample Size by Shuying Liu, Weihong Deng