IUT de Fontainebleau
Département informatique
29/06/2020

RAPPORT DE PROJET:
Détection du port du masque à l'aide d'un réseau de neurones convolutifs

Auteur: Aurélien Peden
Tuteur: Denis Monnerat

SOMMAIRE

SYNTHÈSE

Dans ce rapport se trouve une description détaillée du sujet du projet étant la détection du port du masque à l'aide d'un réseau de neurones convolutifs. Ce projet est une preuve de concept d'application de l'Intelligence artificielle (Deep Learning) à des problématiques concrètes liées à la crise sanitaire due au COVID-19. Plus qu'une preuve de concept liée à la crise actuelle, c'est aussi un projet illustrant les diverses techniques émergentes dans la discipline du Deep Learning, permettant l'augmentation des précisions sur des tâches de reconnaissances d'images par ordinateurs en lien avec différents domaines (imagerie médicale, industrie…).

Il sera décrit le processus et les différentes expérimentations ayant rendu possible la détection du port du masque au sein de flux vidéos, le dataset utilisé, les différentes techniques employées afin d'obtenir un taux de précision de 97% sur les détections ainsi que du choix de l'implémentation de l'algorithme de reconnaissance faciale (Haar cascade) et d'une description de son fonctionnement.

Dans ce rapport se trouvera par ailleurs les ressources utilisées ayant permis l'élaboration de cette preuve de concept, ainsi que des liens vers des rapports de recherches posant les bases des algorithmes et techniques utilisées au sein de ce projet.

Une grande partie de ce projet a été la recherche d'informations et l'apprentissage des connaissances (ainsi que l'étude de rapports de recherche) permettant de réaliser le modèle de prédiction.

PRÉSENTATION DU SUJET

Ce projet est une preuve de concept de détection du port du masque en lien avec la crise sanitaire actuelle due au COVID-19. Le but est d'appliquer les technologies actuelles liés à l'intelligence artificielle et plus précisément au Deep Learning (réseau de neurones ayant plusieurs couches de neurones) à la reconnaissance d'image (et

de données vidéos) permettant de localiser les visages des personnes et de détecter si oui ou non cette personne porte un masque (chirurgical, en tissu…). Un tel concept pourrait être utilisé (voire le sont déjà) dans certaines entreprises, lieux publics (magasins, gares…) afin de sensibiliser les personnes aux gestes barrières mis en places par les différentes institutions (OMS, gouvernements…), les gestes barrières faisant partie des mesures essentielles de lutte contre une pandémie. Ce projet s'inscrit donc dans un cadre de sensibilisation aux gestes barrières.

Le projet a été réalisé avec le langage Python et à l'aide de plusieurs librairies (TensorFlow 2.0 pour la conception du réseau de neurones, Opencv pour la reconnaissance faciale et le travail sur les images, scikit-learn pour le travail sur le jeu de données ainsi que matplotlib pour les graphiques sur la précision des prédictions). Le serveur a été réalisé avec le framework Flask.

RÉALISATION

Jeu de données

Afin de réaliser une intelligence artificielle (modèle de prédiction), il est nécessaire d'utiliser un jeu de données (ou d'en concevoir un) afin que le modèle puisse apprendre à effectuer la tâche qui lui est assigné, en étudiant par lui-même des centaines (voire milliers) d'exemples rattachés à un label (afin de signaler à l'IA à quelle catégorie l'image appartient, c'est-à-dire "masque" ou "aucun masque"). Il a donc été nécessaire d'utiliser un jeu de données contenant des centaines d'images de visages portant (ou ne portant pas) de masque afin que l'intelligence artificielle puisse reconnaître ces deux cas précis, et puisse à l'avenir classifier de nouveaux exemples qu'elle n'a pas encore rencontré auparavant.

Plusieurs problématiques se posent par rapport à ce dernier point, l'une d'entre elles serait que l'IA ait du mal à généraliser ses prédictions sur de nouveaux exemples, ce problème a un nom, "overfitting", une fois l'IA réalisée, elle devra effectuer des prédictions sur des images n'ayant pas de label (les images étant nouvelles et n'ayant donc pas été travaillées par l'homme) avec une bonne précision, qui est de 97% pour ce projet. Cette précision est évaluée sur un jeu de données amputé du "training set" (jeu de données sur lequel le modèle apprends à effectuer la tâche lui étant assignée, ici la reconnaissance du port du masque), c'est-à-dire sur des exemples n'ayant jamais été rencontrés lors de l'entraînement du modèle. Il y a différentes façons de calculer ce pourcentage, cette méthode sera détaillée notamment à l'aide de schémas

dans la partie “conception du réseau de neurones convolutifs”. Il sera par ailleurs mentionné dans cette partie comment le problème de l’overfitting a été évité.

Il a été difficile d’utiliser un jeu de données comportant un nombre convenable de visages portant des masques. Il a donc été décidé d’utiliser un dataset “artificiel”, où des masques sous forme d’image ont été appliqués sur un jeu de données contenant des visages réels. Ce dataset a été conçu par une personne ayant appliqué automatiquement ces faux masques sur des centaines d’images à l’aide d’algorithmes reconnaissant les points clés des visages (nez, oeils, bouche…) afin d’éviter de réaliser un tel travail à la main.
Cependant, même en utilisant de telles techniques, le nombre d’images contenues dans le jeu de données est relativement faible (environ 1300 images, soit environ 600 par catégories). Certaines tâches telles que la reconnaissance des chiffres se basent sur un dataset de 70 000 images. Il a donc été nécessaire d’utiliser des techniques modernes de Deep Learning qui seront mentionnées dans la partie liée à la conception du réseau de neurones.

Exemple d’images du jeu de données utilisé

Reconnaissance faciale

Afin d’effectuer la détection du port du masque, il est auparavant nécessaire de détecter les visages au sein d’une image, afin de les extraire et de les donner par la

suite au réseau de neurones. Afin de réaliser cette tâche, il a été décidé d'utiliser l'algorithme "Haar cascade", datant d'il y a quelques dizaines d'années mais fonctionnant très bien.

L'avantage de l'utilisation d'un tel algorithme plutôt que l'utilisation d'un autre réseau de neurones et le faible coût de performance en termes de calcul. Le fonctionnement de l'algorithme se base sur le principe de "convolution" actuellement utilisé massivement dans les réseaux de neurones liés à la reconnaissance d'image (et portant donc le nom de réseau de neurones convolutifs). Des matrices (tableaux de valeurs) parcours l'image sur laquelle on souhaite détecter les visages, et extraient diverses informations concernant les contours du visage se situant dans l'image. Chacune des matrices correspond à une "feature", une caractéristique permettant de détecter un visage, c'est-à-dire les yeux, le nez, les lèvres… Certaines de ces matrices permettent de détecter spécifiquement les contours (profondeurs, par exemples les yeux), d'autres les lignes (les lèvres, le nez). Ces matrices sont divisées en 2 groupes de valeurs (pixels sombres, ou blancs) et sont superposées sur l'image en question puis la parcourent mettant en relief les caractéristiques d'un visage.

L'image de droite représente le cas idéal de détection des contours (on distingue une nette différence entre la partie claire, les 0 et celle sombre, les 1). Ces chiffres représentent l'intensité des pixels en question. Sur une image, le cas idéal n'est pas atteignable. Une comparaison est donc effectuée entre la matrice parcourant l'image et celle de la caractéristique en question représentant le cas idéal. Plus précisément, une moyenne de l'intensité des pixels de chacune des matrices est faite, puis soustraite l'une à l'autre, donnant un score entre 0 et 1, plus le score est proche de 1, plus il est probable que la caractéristique détectée soit celle d'un visage.

$$\Delta = \text{dark} - \text{white} = \frac{1}{n}\sum_{\text{dark}}^{n} I(x) - \frac{1}{n}\sum_{\text{white}}^{n} I(x)$$

D’autres approches auraient pu être utilisées pour la reconnaissance faciale telle que l’utilisation d’un autre réseau de neurones, mais les calculs effectués par un réseau de neurones sont très coûteux en performances, et sont donc à éviter lorsqu’il est possible de faire autrement.

Une fois les détections effectuées, les coordonnées des visages (un rectangle par visage) sont retournées par l’algorithme, ici implémenté à l’aide de la librairie OpenCV permettant d’effectuer diverses tâches liées à la vision par ordinateur.

```
face_rects = self.face_detector.detect(frame, scaleFactor=1.2,
                                       minNeighbors=3, minSize=(40, 40))

for (x, y, w, h) in face_rects:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

Chacune des coordonnées (appelées tuples dans le langage Python) appartenant aux visages sont ensuite itérées au sein d’une boucle, et un rectangle est dessiné sur l’image en utilisant les coordonnées en question.

Les paramètres de la fonction sont la taille minimale des visages devant être acceptés comme étant bien un visage (minSize), le nombre de voisins (en lien avec les matrices précédemment mentionnées, un nombre important donne des détections plus précises, mais certains visages pourraient donc être ignorés), et le scaleFactor (certains visages peuvent être en arrière-plan, d’autre au premier plan, il faut donc préciser à l’algorithme comment s’adapter à une telle situation).

Une fois la partie graphique de la détection réalisée, le visage est extrait de l’image d’origine pour la fournir au réseau de neurones sur laquelle la prédiction sera réalisée (on ne peut pas fournir directement l’image d’origine, le réseau de neurones fonctionnant qu’avec des images de tailles précises, ici de taille 224x224 pixels). L’extraction se déroule d’une façon similaire au dessin du rectangle de détection. L’image étant simplement une matrice, on sélectionne les pixels nous intéressant (donc les coordonnées du visage détecté par l’algorithme).

```
face_image = frame[y:y+h, x:x+w]
```

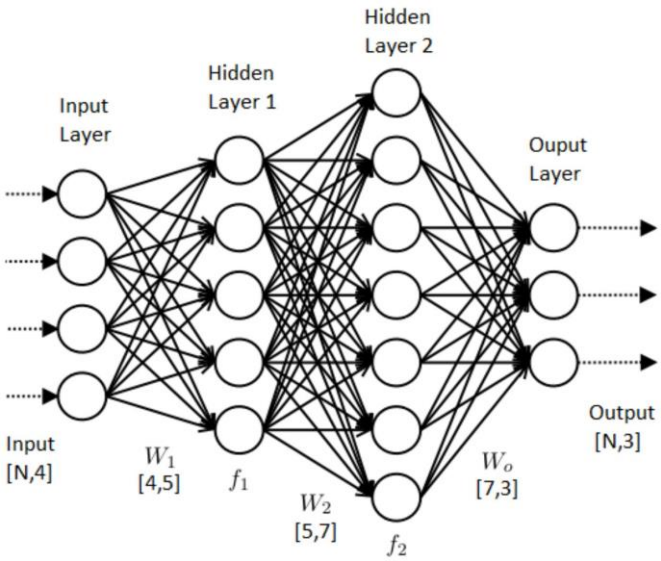
Conception du réseau de neurones convolutif

La conception du réseau de neurones convolutifs est le cœur de ce projet, et la partie la plus complexe en terme de théorie. Un réseau de neurones est un système

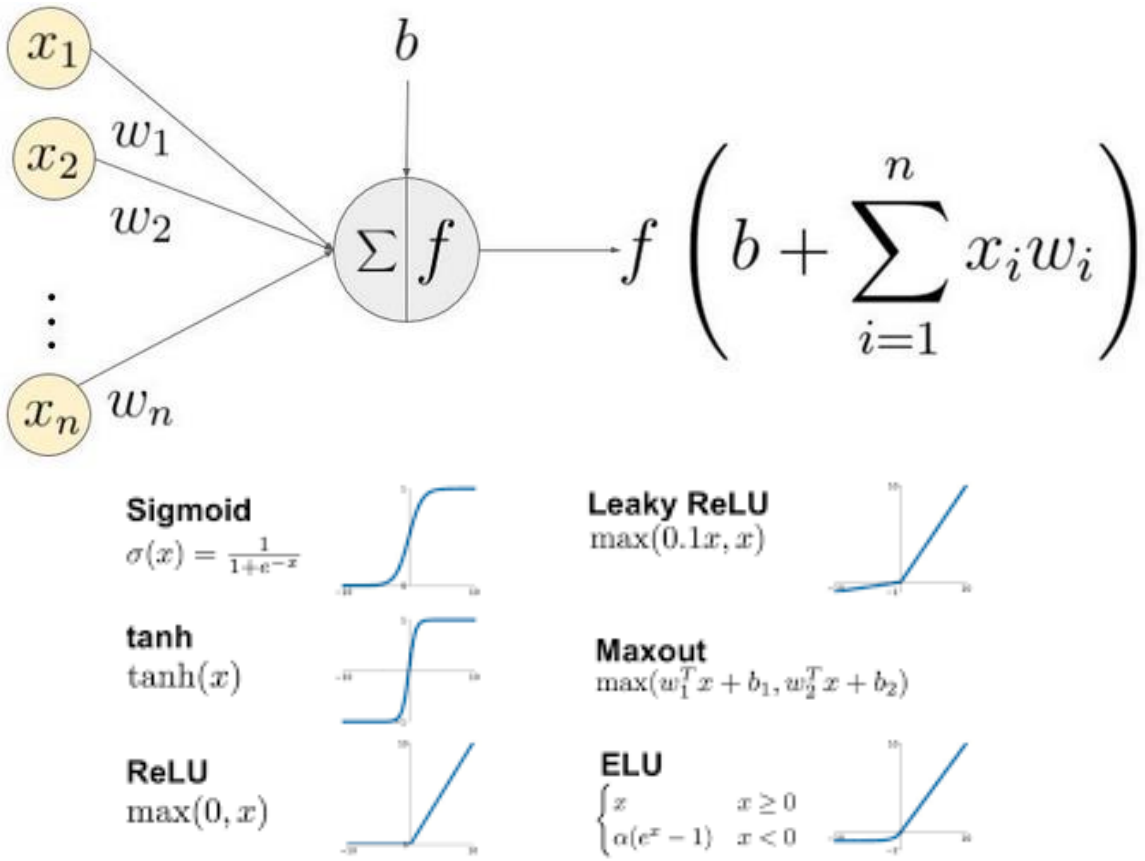
complexe inspiré du fonctionnement des neurones du cerveau humain, les neurones communiquent entre eux à l'aide de signaux chimiques lorsque ces neurones en question s'activent. Un réseau de neurones artificiel suit le même principe, des données sont fournies aux neurones, et ces neurones s'activent à la suite de calculs, et transmettent ainsi les valeurs calculées aux neurones de la couche suivante.

Ces calculs sont effectués sur des valeurs issues du jeu de données sur lequel le réseau de neurones est entraîné, ici, le problème étant la reconnaissance d'image, ces valeurs sont les pixels des images fournies au réseau de neurones. Chacune de ces valeurs sont associé à un "poids", représentant l'importance de cette valeur pour mener à bien la tâche que l'on souhaite accomplir (ici détecter le port du masque). Ce sont donc ces valeurs qui seront modifiées tout au long de l'entraînement du réseau de neurones. Le réseau de neurones apprend effectivement à optimiser les poids (trouver la meilleure valeur possible) permettant d'effectuer une prédiction la plus précise possible. Par soucis de performance, des techniques telle que la vectorisation sont employées, permettant d'éviter des boucles imbriquées lors des calculs entre les valeurs (pixels) et les poids. Les poids sont donc représentés par des matrices, de même pour les valeurs du jeu de données.

Sur le schéma suivant d'un réseau de neurones classique, on peut voir que celui-ci est constitué de 3 couches : 2 couches "cachées" et une de sortie (output). La première couche étant les neurones acceptants chacun d'entre eux une valeur par image. Pour ce projet, le réseau de neurones accepte des images de 224x224, il y a donc 224*224=50 176 neurones à l'entrée du réseau, une par pixel.



Chaque neurone réalise un calcul, avec pour sortie une valeur entre 0 et 1. Ce calcul est réalisé par une fonction d'activation (similaire à l'analogie avec les neurones biologiques mentionnées précédemment). Cette fonction prend en paramètres la multiplication des matrices de poids (w pour weights) et la matrices des données (x, représentant un exemple du jeu de données).



Synthèse des différentes fonctions d'activation existantes à ce jour

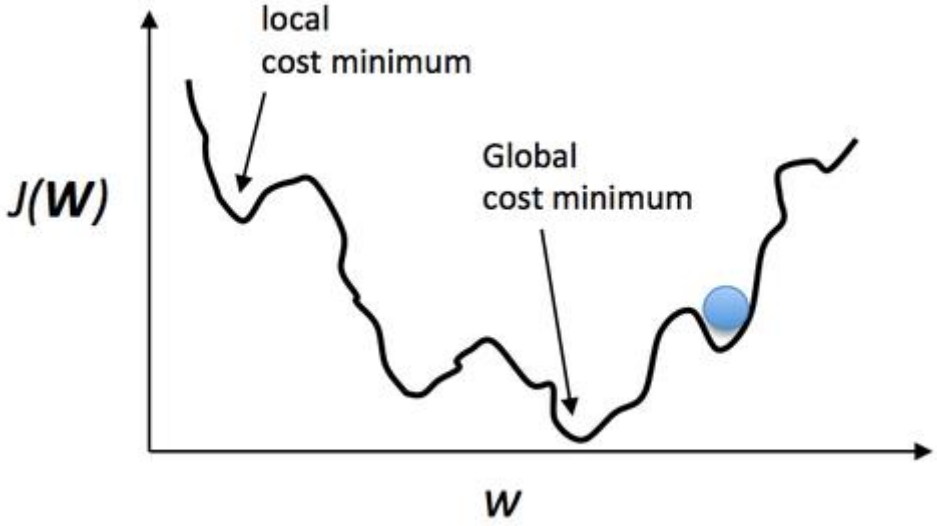
Ces valeurs sont ensuite transmises aux couches suivantes, jusqu'à atteindre la couche de sortie. Un score est ensuite calculé pour mesurer la proximité de la prédiction avec le label réel associé à l'exemple en question.

$$Loss(y,\hat{y}) = \sum_{i=1}^{n} (y - \hat{y})^2$$

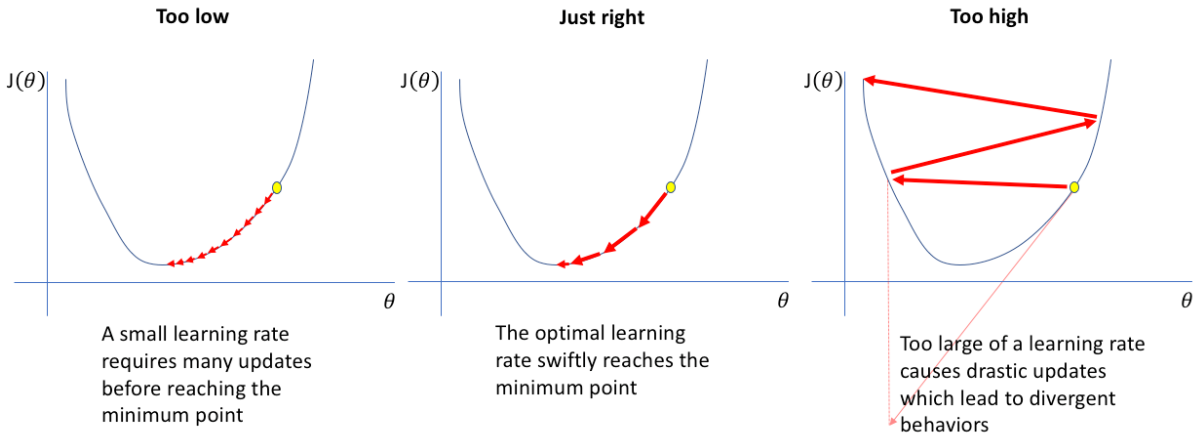
Dans cette équation, y représente le label et y^ représente la valeur calculée en sortie du réseau de neurones.

A la suite de ce calcul, le réseau de neurones a pour but d'optimiser le coût du réseau, c'est à dire de réduire le plus possible la fonction Loss, et donc de modifier les matrices de poids afin de réduire l'écart entre la valeur prédite et celle du label. C'est donc un processus itératif, les données passent chacune à travers le réseau une fois par étape (appelée epoch), chaque étape permettant à nouveau de calculer la fonction de coût, et donc de modifier la matrice de poids. On appelle cela l'algorithme de backpropagation. La fonction de coût est dérivée et propagée de la sortie du réseau de neurones vers l'entrée, afin d'optimiser chacun des poids et de "chercher" la meilleure optimisation possible afin de minimiser la fonction de coût.

En termes mathématiques, on parle alors de dérivées partielles et de problème d'optimisation de fonctions convexes. Le but est de trouver le minimum global de la fonction de coût, permettant de réaliser des prédictions aussi précises que possibles. Pour cela, on soustrait aux poids la valeur résultant de la dérivation de la fonction de coût, multipliée par un scalaire, une valeur, le "learning rate", c'est-à-dire la vitesse à laquelle l'optimisation (représentée ici par le point bleu descendant vers le minimum global) est réalisée.



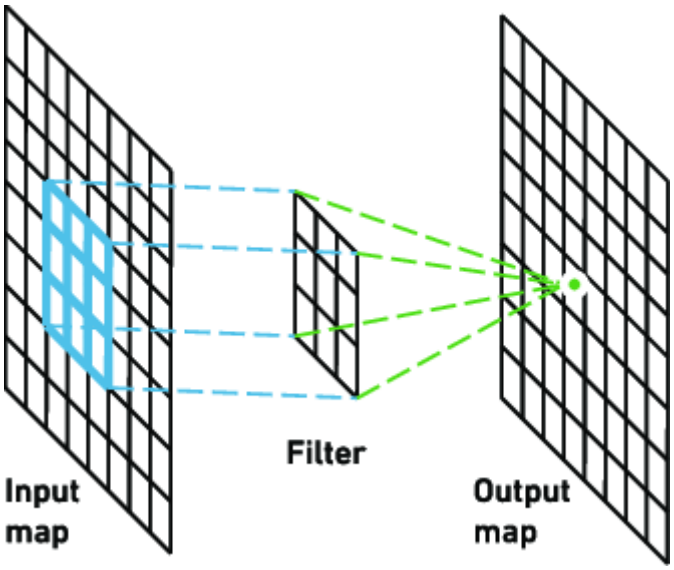
Le learning rate constitue l'un des paramètres à ajuster lors de la conception du réseau de neurones. Un learning rate trop élevé causerait au point bleu le fait de ne jamais atteindre le minimum global, un learning rate trop faible causerait un temps de calcul beaucoup trop long comme le montre le schéma suivant. Toutes ces étapes constituent ce que l'on appelle la "descente de gradient", le coût descend peu à peu vers le minimum global (point de la courbe le plus bas) grâces aux dérivées partielles calculées lors de la backpropagation.



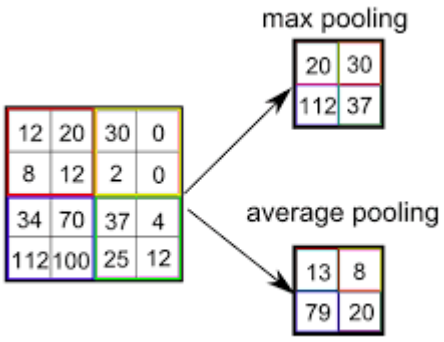
Pour ce projet, il est question de réaliser de la reconnaissance d'image. Il faut donc utiliser un réseau de neurones spécifique, appelé CNN (convolutionnal neural network), ou réseau de neurones convolutifs. Un réseau de neurones convolutifs suit le même principe que le réseau précédemment décrit, sa particularité tient dans le

type de neurones qu'il contient. Il est en effet constitué de neurones de convolutions effectuant des opérations complexes sur les matrices de valeurs (et donc images) et produit en sortie des filtres (dont le nombre est choisi lors de la conception du réseau, et ce pour chaque couche de convolution).

Chaque filtre apprend à détecter des caractéristiques spécifiques de l'image (à la manière de l'algorithme haar cascade) après l'avoir parcourue à la manière d'une fenêtre glissant sur l'image, et transmet ces filtres à la prochaine couche.



Cependant, un problème se pose, ces neurones augmentent drastiquement le nombre de valeurs à calculer au fil des couches, car chaque filtre donne une autre image en sortie. Il y a donc une image de plus par filtre. Comme vu précédemment, le temps d'entraînement d'un réseau de neurones est un problème important à ne pas ignorer. Il y a donc une autre opération à effectuer à la suite d'une convolution, opération dites de "pooling". Cela consiste à résumer les filtres (et donc caractéristiques apprises) en réduisant la taille des images obtenues à la suite de ces filtres. La plus populaire est l'opération de max pooling. Une fenêtre de taille choisie (souvent 2x2 comme sur le schéma suivant) parcours l'image obtenue à l'aide du filtre en retenant seulement la valeur maximale se situant dans la fenêtre. Cela permet de diviser par 2 la taille de l'image en question, et ainsi de diviser par 2 le nombre de valeurs à calculer, augmentant alors la performance du réseau de neurones.



Toutes ces techniques entraînent un nombre de calculs massif à réaliser. Il a donc été décidé d'utiliser une technique dite de "Transfer Learning", permettant de réutiliser un réseau de neurones déjà entraîné sur une autre problématique, plus précisément sur le jeu de données "ImageNet", populaire dans le milieu de la recherche académique. Cela permet d'éviter plusieurs milliers de calculs de poids. Cependant la tâche à réaliser pour ce projet est la détection du port du masque, il faut donc que le réseau de neurones puisse apprendre à reconnaître ce type d'image. Il est donc nécessaire de rajouter par-dessus le réseau de neurones que l'on souhaite utiliser, des couches avec des poids devant être appris par le nouveau réseau de neurones. Le réseau déjà entraîné sert donc de base, sur laquelle il va être rajouté nos propres couches, afin d'adapter ce réseau à la tâche que l'on souhaite accomplir.

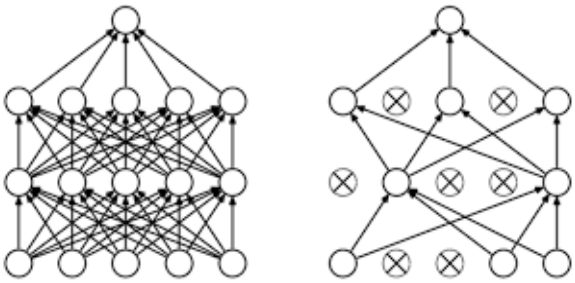
```
base_model = MobileNetV2(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

head_model = base_model.output
pool = AveragePooling2D(pool_size=(7, 7))(head_model)
flatten = Flatten(name="flatten")(pool)
dense = Dense(128, activation="relu")(flatten)
dropout = Dropout(0.5)(dense)
output = Dense(2, activation="softmax")(dropout)

model = Model(inputs=base_model.input, outputs=head_model)
```

L'architecture (l'ensemble de couches formant le réseau de neurones) utilisée pour le modèle de base est appelé "MobileNetv2". Il est ajouté sur ce réseau 3 couches, une de pooling (Average pooling), une couche classique "Dense" de 128 neurones, ainsi qu'une autre de 2 neurones, fournissant chacune d'entre elle une probabilité correspondant à chaque catégorie (masque détecté ou non). Les fonctions d'activation utilisées sont "relu, rectified linear unit" et "softmax" pour le calcul de la probabilité. On peut voir l'utilisation de deux autres opérations, une appelée "Flatten", permettant de transformer les images obtenues à la suite des convolutions en un seul vecteur de pixels (on aplatit l'image), et une opération appelée "Dropout" permettant d'éviter le problème d'overfitting, c'est-à-dire la difficulté pour le réseau à généraliser les prédictions sur des exemples encore jamais rencontrés.

On appelle cela une technique de régularisation. Cette technique vise à impacter les calculs des poids/fonctions de coût afin d'éviter le problème décrit précédemment. La technique de dropout vise à ignorer l'activation d'un certain pourcentage (ici 50%, pour 0,5) de neurones de la couche en question. Cela permet au réseau de neurones d'éviter d'apprendre par cœur les exemples rencontrés et de ne pas savoir effectuer de prédictions sur de nouveaux exemples, comme si le réseau apprenait sa leçon par cœur.



(a) Standard Neural Net (b) After applying dropout.

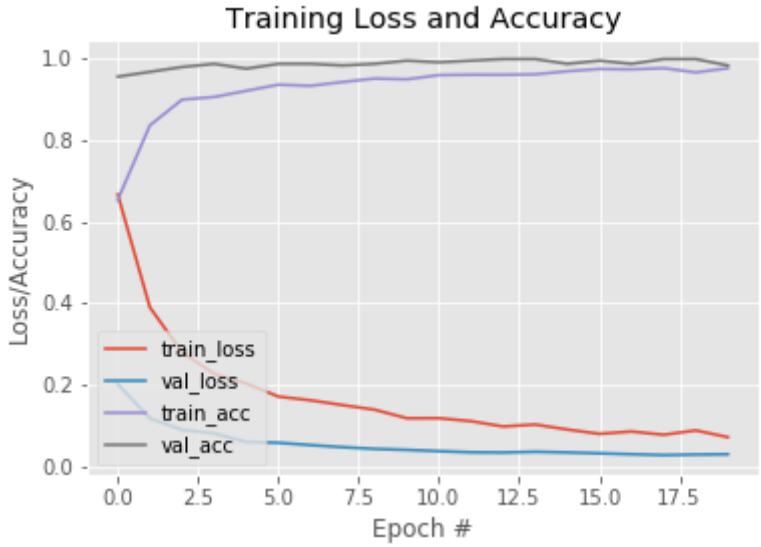
Une autre façon d'éviter le problème d'overfitting est de fournir un nombre plus important d'exemples au réseau de neurones. Cependant comme mentionné précédemment, dû à la nature récente de ce type de tâche, à savoir la détection du port du masque, le nombre de données disponible est relativement faible. Il a donc été nécessaire d'utiliser une technique de "Data augmentation", c'est-à-dire de créer de nouvelles données à partir de celles que l'on dispose actuellement. Pour cela on peut réaliser diverses opérations sur l'image, inverser son sens, l'incliner légèrement, effectuer un zoom, la décaler horizontalement, verticalement… C'est ce qui est fait avec la ligne de code suivante.

```
train_generator = ImageDataGenerator(rotation_range=20, zoom_range=0.15, width_shift_range=0.2, height_shift_range=0.2, shear_range=0.15,
horizontal_flip=True, fill_mode="nearest")
```

Le nombre d'images disponible à l'origine est donc démultiplié par toutes ces opérations, permettant au réseau de neurones de disposer d'un nombre de données plus important sur lequel s'entraîner et se perfectionner.

D'autres techniques auraient pu être utilisées, telle que les GANs (generative adversarial networks), un autre type de réseau de neurones permettant de générer de nouvelles images à partir d'exemples. Cependant cette technique est encore très récente (~2015) et n'est donc pas optimisé pour un tel problème par son manque de maturité.

Toutes ces techniques ont permis d'obtenir une précision de 97-98% sur le test set (jeu de données n'ayant jamais été donné au réseau de neurones et permettant d'évaluer les performances de ce dernier, représenté par la courbe violette). C'est-à-dire que 98% des données ont été correctement prédites.



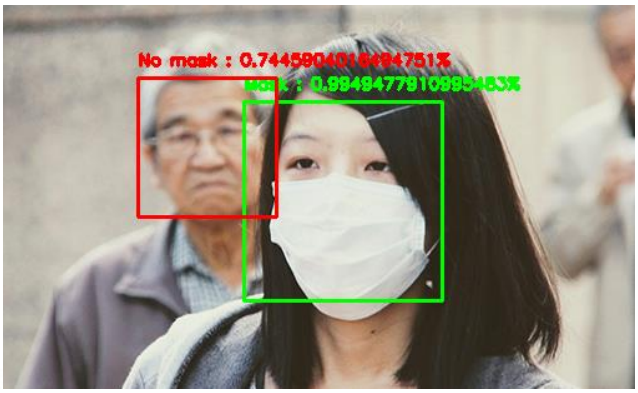

```
   print(classification_report(test_y.argmax(axis=1), predictions, target_names=lb.classes_))

precision    recall  f1-score   support

   with_mask       0.99      0.96      0.98       138
without_mask       0.96      0.99      0.98       138

    accuracy                           0.98       276
   macro avg       0.98      0.98      0.98       276
weighted avg       0.98      0.98      0.98       276
```

Score par catégorie



Détection du port du masque sur un flux vidéo

Une fois la conception du réseau de neurones finalisée, il a fallu “relier” la partie modèle de prédiction au flux vidéo. Dans un premier temps, il a été décidé d’effectuer les prédictions sur de simples images afin de tester les performances du modèle. Pour cela, la librairie OpenCV a été utilisé afin de charger les images.

L’image chargée est ensuite redimensionnée si sa taille est trop grande afin de ne pas impacter les performances de la reconnaissance faciale, fonctionnant moins bien avec des images de taille trop élevée. L’algorithme haar cascade nécessitant des images en noir et blanc, celle-ci est donc convertie afin d’être compatible avec l’algorithme.

```
ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", required=True,
                help="path to where the face cascade resides")
ap.add_argument("-i", "--image", required=True,
                help="path to where the image file reside")
args = vars(ap.parse_args())

image = cv2.imread(args["image"])
(h, w) = image.shape[:2]
if h > 1080:
    image = cv2.resize(image, (600, 800))
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Une fois l’image convertie et transmise à l’algorithme, ce dernier, comme vu précédemment, renvoie les coordonnées des visages sur l’image. Les visages sont ensuite extraits de l’image d’origine, convertie dans la dimension 224x224 pour la rendre compatible avec les neurones d’entrée du réseau, puis chaque pixel est divisé par 255.0 (une valeur rgb étant comprise entre 0 et 256), permettant ainsi l’image d’avoir des valeurs entre 0 et 1. Ce processus est appelé “normalisation”, et a pour but de faciliter les calculs pour le réseau de neurones, car les plages de valeurs sont ainsi restreintes.

Chaque visage est ensuite transmis au modèle pour effectuer les prédictions, et ce dernier retourne une probabilité par catégorie (grâce à la fonction d’activation softmax). Les deux probabilités sont ensuite comparées, et l’affichage est adapté en fonction de la prédiction (cadre rouge si aucun masque n’est détecté, vers dans le cas contraire). À la différence de l'entraînement du réseau de neurones s’effectuant avec une carte graphique, les prédictions sont réalisées avec un processeur, limitant ainsi l’infrastructure nécessaire pour héberger un tel serveur (l’utilisation d’une carte graphique entraînera la location d’un serveur à un taux horaire).

```
face_images = []
for (x, y, w, h) in faceRects:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

    face_image = image[y:y+h, x:x+w]

    face_image = cv2.cvtColor(face_image, cv2.COLOR_BGR2RGB)
    face_image = cv2.resize(face_image, (224, 224))

    face_image = face_image/255.0

    face_image = img_to_array(face_image)
    face_image = np.expand_dims(face_image, axis=0)

    face_images.append(face_image)

    (mask, without_mask) = model.predict(face_image)[0]

    if mask > without_mask:
        print("Mask! :", mask)
        color = (0, 255, 0)
        cv2.putText(image, "Mask : {}%".format(mask), (x, y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
    else:
        color = (0, 0, 255)
        cv2.putText(image, "No mask : {}%".format(without_mask), (x, y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
        print("without mask ! : ", without_mask)
```

La détection au sein d'un flux vidéo suit exactement le même principe, car une vidéo est juste une succession d'images. Il suffit pour cela de remplacer le chargement d'une image par le lancement d'un flux de capture vidéo, avec la ligne suivante.

```
stream = cv2.VideoCapture(0)
```

Conception de l'application web

La dernière fonctionnalité de ce projet réside dans l'application web, permettant de tester cette preuve de concept directement au sein d'un navigateur internet. Pour cela, le framework flask a été utilisé pour héberger les précédentes parties du programme (capture vidéo, prédictions…), hormis le programme de la conception du réseau de neurones, le modèle étant simplement enregistré dans un fichier une fois

conçu. Le serveur s'occupe donc de charger le modèle à son démarrage, et lorsqu'une connexion au site internet est effectuée, la capture vidéo est lancée. L'utilisateur peut donc voir en direct les prédictions directement sur le site, grâce à sa webcam.

```
@app.route('/')
def index():
    return render_template('index.html')


def gen(camera):
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
```

Une fois la page "index.html" chargé, le serveur s'occupe de récupérer chaque image de la capture vidéo à l'aide de la fonction get_frame, puis la retourne au html suivant, permettant de voir le flux vidéo au milieu de la page internet.

```
<main>
    <img id="bg" class="center" src="{{ url_for('video_feed') }}">
</main>
```

La fonction get_camera contient le code des précédentes parties, dans une classe "VideoCamera" lui étant dédié et dont le constructeur est le suivant.

```
class VideoCamera(object):
    def __init__(self):
        self.video = cv2.VideoCapture(0)

        self.model = load_model('../face_mask.model')
        self.face_detector = FaceDetector(
            './haarcascade_frontalface_default.xml')
```

Conclusion

En conclusion, ce projet rassemble donc les dernières techniques d'apprentissage profond (Deep Learning), constituant le cœur des dernières avancées dans le domaine de l'intelligence artificielle. Il démontre par ailleurs la possibilité d'effectuer de nouvelles tâches avec un nombre de données relativement faible grâce aux techniques de Transfer Learning. Ce genre de concept, si bien utilisé pourrait s'avérer être un atout majeur pour l'application des gestes barrière et le recul de la pandémie actuelle, mais il peut aussi poser des problèmes au niveau éthiques, notamment vis-

à-vis de la surveillance de masse. C'est pourquoi il est important de démocratiser l'utilisation de ces technologies.

Plusieurs améliorations auraient pu être réalisées, notamment la détection des visages à l'aide d'un autre réseau de neurones, détectant le centre d'un visage, puis sa largeur et sa hauteur, plutôt que d'utiliser un algorithme tel qu'haar cascade. Cependant cette solution entraînerait un coût en termes de calculs plus important.

Un cheminement similaire à ce projet pourrait être utilisé pour d'autres tâches, comme la lecture automatique de plaques d'immatriculation, la reconnaissance de la langue des signes, ou la détection d'anomalie sur des données d'imagerie médicale.

D'un point de vue personnel, ce projet m'a permis d'appliquer diverses connaissances en Deep Learning que j'ai étudié en autodidacte à l'aide de certifications mentionnées dans la partie suivante. J'ai par ailleurs pu apprendre le framework OpenCV, la reconnaissance faciale ainsi que la mise en production d'un réseau de neurones. Cela m'a aussi permis de me préparer aux enseignements d'école d'ingénieurs concernant la science des données, domaine dans lequel je souhaite me spécialiser.

Ressources

Rapport de recherche utilisés

- Haar cascade
  - https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
- Visualizing and understanding convolutionnal neural networks:
  - https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf
- Dropout : A simple way to prevent neural network from overfitting
  - https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Dataset

- Face mask dataset
  - https://github.com/prajnasb/observations/tree/master/experiements/data

Cours

- CS231n : Convolutional Neural Networks for Visual Recognition
  - http://cs231n.stanford.edu/
- Machine Learning by Stanford on Coursera:

    - https://www.coursera.org/learn/machine-learning
- Deeplearning.ai specialization on Coursera:
    - https://www.coursera.org/specializations/deep-learning
- Livre hands-on machine learning d'Aurélien Géron
- Documentation OpenCV
    - https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html