SIMPLON.CO

Projet N°14
Avril 2022

RAPPORT

Modèle intelligent pour la détection des masques

Transfer learning pour la détection des masques

Auteurs : **Maïna LE DEM**

Indice: A
Date d'émission: 05/04/2022

SOMMAIRE

Partie 0 : Veille Transfer Learning

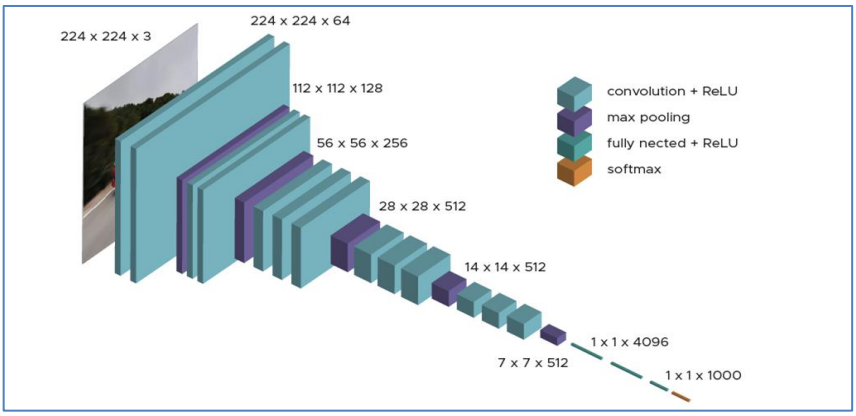
0.1 - *Architecture VGG16*

Définition : VGG est un réseau de neurones convolutionnels

Il existe deux algorithmes disponibles : VGG16 et VGG19 (VGG19 présente un plus grand nombre de couches de convolution)

VGG est un algorithme connu en Computer Vision très souvent utilisé par transfert d'apprentissage pour éviter d'avoir à le réentraîner et résoudre des problématiques proches sur lesquelles VGG a déjà été entraîné.

Démonstration :

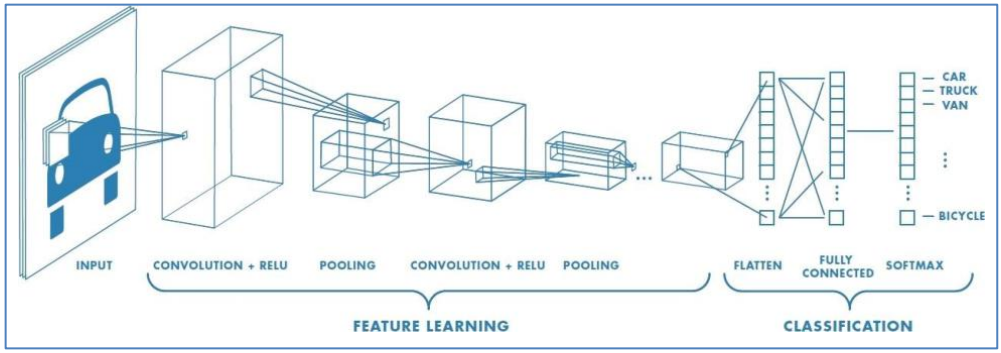


0.2 - *Transfer Learning*

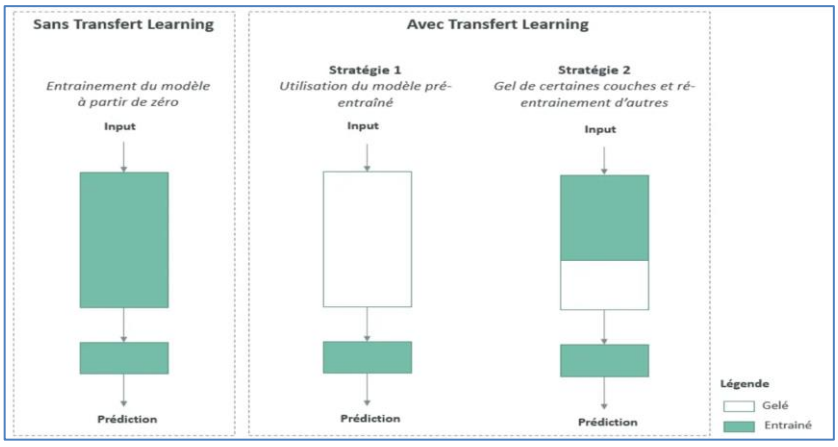
Application aux problématiques de Deep Learning, 2 types de stratégies :

- Utilisation de modèles pré-entraînés comme extracteurs de features
- Ajustement de modèles pré-trainés

On récupère la partie du modèle existant pour extraire les caractéritiques, lui donner nos propres images, puis créer nos propres couches à la fin pour faire la classification



Il permet donc d'obtenir de meilleures performances avec un temps d'entraînement plus court : Approche de Transfer Learning en Deep Learning



3 questions, pouvoir choisir la stratégie la plus adaptée :

- Que veut-on transférer ?
- Quand peut-on transférer ?
- Comment va-t-on réaliser le transfert ?

0.3 - *Data augmentation*

Améliore rapidement son modèle de Deep Learning.

Pour générer des images supplémentaires quand on manque de données

0.4 - *Comment sauvegarder le meilleur modèle sur Keras*

Création d'un point de sauvegarde : `tf.keras.callbacks.ModelCheckpoint(save_weights_only=True save_best_only=True)`

Enregistrement dans un fichier `*.h5`

0.5 - *Comment sauvegarder l'historique de l'apprentissage sur Keras*

Création d'un point de sauvegarde : `tf.keras.callbacks.ModelCheckpoint(save_weights_only=True)`

0.6 - *Overfitting Underfitting*

Le sur-ajustement (overfitting) et le sous-ajustement (underfitting) sont deux des pires problèmes dans le Machine Learning.

L'Overfitting (sur-apprentissage) désigne le fait que le modèle prédictif produit par l'algorithme de Machine Learning s'adapte bien au Training Set, le modèle prédictif capturera les corrélations généralisables et le bruit produit par les données.

L'Underfitting (sous-apprentissage), sous-entend que le modèle prédictif généré lors de la phase d'apprentissage, s'adapte mal au Training Set, le modèle prédictif n'arrive même pas à capturer les corrélations du Training Set. Par conséquent, le coût d'erreur en phase d'apprentissage reste grand.

Partie 1 : Base de données, Analyse et Préparation

1.1 - *Importation des connées, création du jeu*

- Import des images avec OpenCV dans une boucle for
- Redimensionnement et standardisation des images
- Création de la target, avec 'with_mask'=1, 'without_mask'=0

1.2 - *Splitter les données en données d'apprentissage, validation et test*¶

- Séparation du jeu de donnée en train et test
- Création du jeu de validation avec le jeu train
- Standardisation de la target y, après séparation

Partie 2 : Architecture CNN sur Tensorflow

2.1 - *Data Augmentation sur les données d'apprentissage sur mon train*

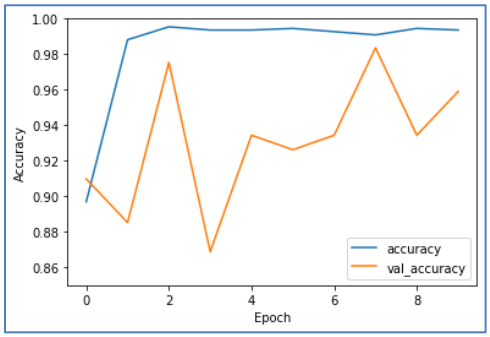
```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)

datagen.fit(X_train)
```

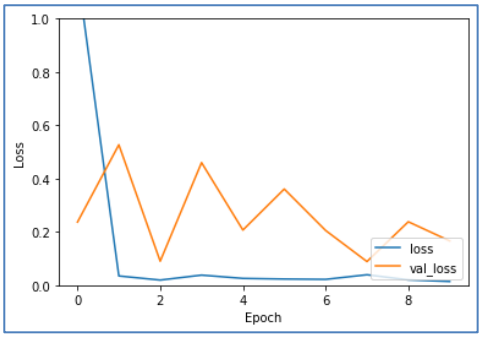
2.2 - *Apprentissage de CNN.*

- Modèle VGG16

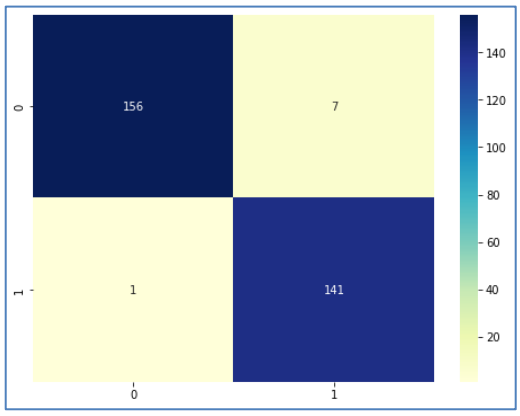
```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freezer les couches du VGG16
for layer in base_model.layers:
    layer.trainable = False
```

- Ajout des couches denses sur le dessus des couches de convolution du VGG16 (flatten et dense)
- Compiler le modèle
- Entraîner le modèle sur le train augmenté (flow permet une transformation),
  En sauvegardant le modèle de la meilleure époque avec « checkpoint »,
  Et la validation avec le jeu de validation
- Évaluer le modèle :
  - Accuracy = 96,35%
  - Loss = 13,33%





- Contrôle sur le jeu d'essai
- Affichage des résultats et de la matrice de confusion
  - Précision du modèle : 0.9737704918032787



2.3 - *Conclusion :*

Le modèle ne semble pas être en overfitting

Partie 3 : Application de détection des masques

- **Test du modèle développé sur des nouvelles images, en affichant un message : Avec masque ou Non masque sur l'image**
  - Prédiction sur 1 photos
  - Prédiction sur plusieurs photos :
- **Code python qui active la Webcam et identifie si la personne porte un masque en affichant le message sur l'image**
  - Dans mon notebook :

```
from detection_mask import appli_tps_reel
appli_tps_reel(loaded_model)
```

  - Dans mon fichier python :

```
liste_label = ['sans masque', 'avec masque']

def appli_tps_reel(loaded_model):
    vid = cv2.VideoCapture(0)

    # Condition/boucle while que tant que la viéo capture
    while(True):
        ret, frame = vid.read()

        imgr = cv2.resize(frame,(224,224), interpolation = cv2.INTER_AREA)
        img_exp = np.expand_dims(np.array(imgr)/255.0, axis=0)

        prediction = loaded_model.predict(img_exp)
        resultat = liste_label[np.argmax(prediction)]

        image_pred = cv2.putText(frame, resultat, (120, 70),
                                 cv2.FONT_HERSHEY_SCRIPT_COMPLEX, 2, (255,0,0))

        # Affiche le résultat
        cv2.imshow('frame', image_pred)

        # the 'q' button is set as the,
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    vid.release()
    cv2.destroyAllWindows()
```