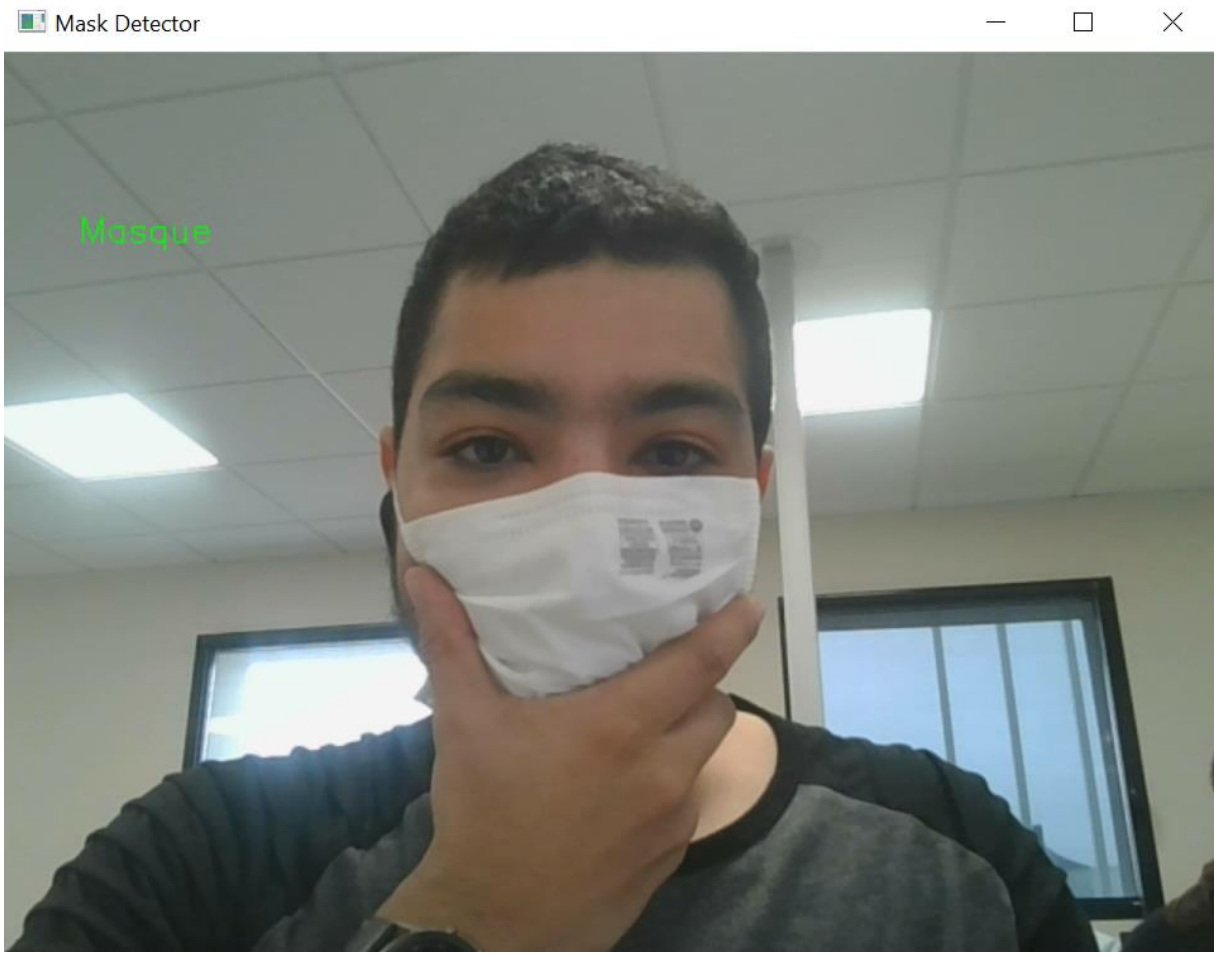

Modèle I.A. pour détection de masque en temps réel

TRANSFERT LEARNING ET RESEAUX DE CONVOLUTION

BOUCLY Kévin, TANGUY Franky | Compte-rendu de brief | 04/04/2022

Table des matières

1. Rappels théoriques

1.1 Le CNN (convolutionnal neural network)

Le réseau de neurone convolutionnel est un type de réseau de neurone dont l'architecture est optimisée dans le traitement d'image.

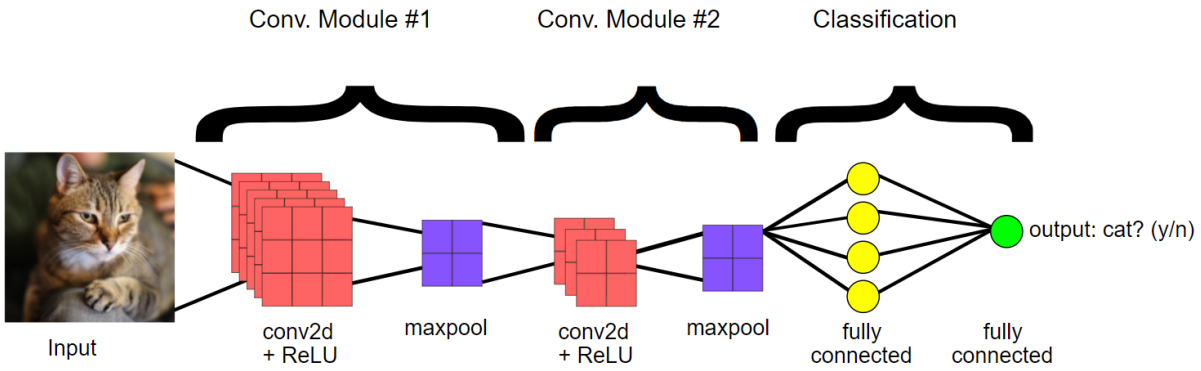

Figure 1 : Architecture d'un CNN

Le réseau va appliquer successivement un filtre de convolution sur l'image, qui va progressivement permettre d'extraire des features de l'image, puis une opération de pooling pour réduire la dimension de la donnée. Cette succession est répétée autant de fois que nécessaire. On a ensuite une ou plusieurs couches où tous les neurones sont connectés entre eux. En sortie, on a le label souhaité.

1.2 L'architecture VGG16

Le modèle VGG16 est un CNN constitué de 16 couches qui a été entraîne par le dataset ImageNet, un dataset dédié au travail sur la vision par ordinateur. Il contient des millions de photos avec label. Le VGG16 est capable de reconnaître 1000 objets différents en ayant été entraîné avec ces données.

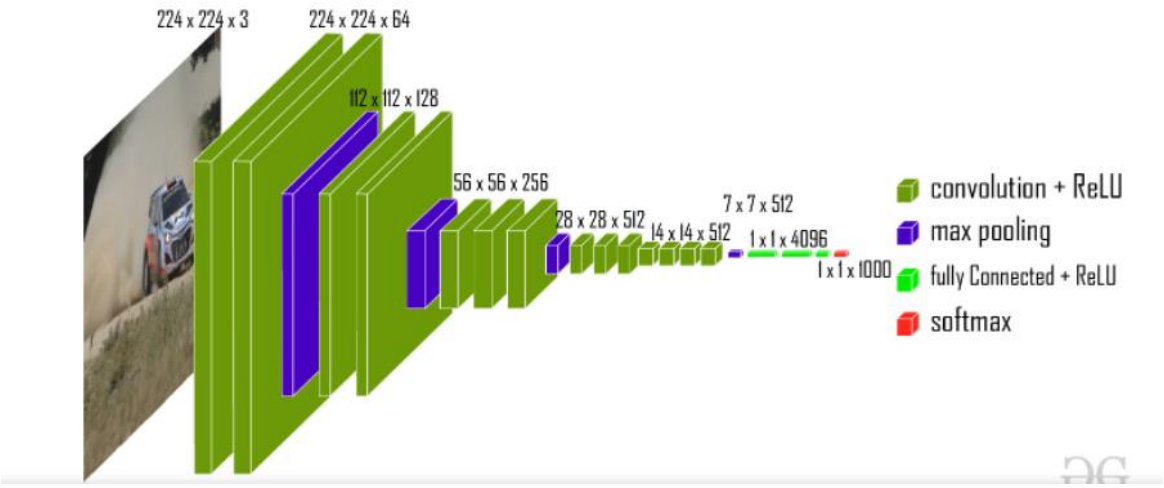

Figure 2 : Architecture du VGG16

Il prend en entrée des images encodé en RGB de dimension 224 x 224. Sa couche de sortie compte 1000 neurones, ce qui signifie que le réseau est entraîné à reconnaître 1000 objets définis. Ce n'est pas notre objectif, et c'est pourquoi nous allons utiliser le VGG16 mais avec uniquement 2 sorties.

```
from keras.applications.vgg16 import VGG16
model = VGG16()
print(model.summary())
```

Figure 3 : Importation du VGG16

1.3 Le Transfert Learning

Nous aimerions utiliser la puissance du modèle VGG16, mais nous n'avons pas la puissance de calcul nécessaire pour recalculer tous les poids adaptés à notre application. Nous allons donc faire ce qu'on appelle du Transfert Learning. On part du constat que le VGG16 est déjà entraîné, donc à priori, au travers de ses couches, il a appris à traiter de l'image et à reconnaître certains schémas.

Le principe est de figer l'apprentissage du VGG16 et de ne reprogrammer que ses dernières couches (celles où les neurones sont tous interconnectés).

2. Programmation du réseau avec Transfert Learning

2.1 Préparation du modèle

```
# 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
```

Figure 4 : Blocage des premières couches du VGG16

Les couches de bases ne sont plus entraînable.

```
# Ajout des nouvelles couches
model = Sequential()
model.add(base_model) # Ajout du modèle VGG16
model.add(Flatten())
# print(model.summary())
model.add(Dense(4096, activation='relu'))
model.add(Dense(4096, activation='relu'))
model.add(Dense(2, activation='softmax'))
```

Figure 5 : Programmation des nouvelles couches

Nous imprimons un résumé de notre réseau de neurones :

```
 Layer (type)                 Output Shape              Param #
=================================================================
 vgg16 (Functional)           (None, 7, 7, 512)         14714688

 flatten (Flatten)            (None, 25088)             0

 dense (Dense)                (None, 4096)              102764544

 dense_1 (Dense)              (None, 4096)              16781312

 dense_2 (Dense)              (None, 2)                 8194

=================================================================
Total params: 134,268,738
Trainable params: 119,554,050
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 6 : Résumé de notre réseau adapté

Nous avons modifié le nombre de sortie de la dernière couche (il est passé de 1000 à 2).

Il est maintenant temps de réentraîner les dernières couches de notre modèle. Le temps de calcul est long, donc nous configurons un espace de stockage pour le meilleur modèle :

```
checkpoint_filepath = './Mask.h5'
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
```

Figure 7 : Stockage du modèle

Pour la suite nous n'aurons plus qu'à le charger avec les lignes de commandes suivantes :

```
model = keras.models.load_model('./Mask.h5')
```

Figure 8 : Chargement du modèle entraîné

2.2 Data augmentation

Afin d'avoir plus de données et de limiter les risques d'overfitting du modèle, nous allons utiliser une technique dite de Data augmentation. Il s'agit de déformer légèrement les images existantes et d'ajouter ces images modifiées aux données d'entraînement. Ainsi, le modèle sera entraîné sur plus de variations d'images et pourra plus facilement détecter un masque.

```
datagen = 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)
```

Figure 9 : Augmentation des données

2.3 Entraînement du modèle

Nous entraînons ensuite le modèle.

```
history=model.fit(datagen.flow(x_train, y_train, batch_size = 32), epochs=10,
validation_data=(x_val,y_val), callbacks=[checkpoint])
```

Figure 10 : Entraînement du modèle

Le calcul est long. Le modèle final a un poids total d'environ 1,4 Go.

```
31/31 [==============================] - 376s 12s/step - loss: 7.2214 - accuracy: 0.8442 - val_loss: 2.2108 - val_accuracy:
0.9793
Epoch 2/10
31/31 [==============================] - 430s 14s/step - loss: 0.0693 - accuracy: 0.9792 - val_loss: 4.3509 - val_accuracy:
0.9668
Epoch 3/10
31/31 [==============================] - 382s 12s/step - loss: 0.0703 - accuracy: 0.9782 - val_loss: 3.2739 - val_accuracy:
0.9710
Epoch 4/10
31/31 [==============================] - 338s 11s/step - loss: 0.0505 - accuracy: 0.9813 - val_loss: 0.8485 - val_accuracy:
0.9876
Epoch 5/10
31/31 [==============================] - 397s 13s/step - loss: 0.0931 - accuracy: 0.9740 - val_loss: 1.8202 - val_accuracy:
0.9834
Epoch 6/10
31/31 [==============================] - 388s 13s/step - loss: 0.0155 - accuracy: 0.9927 - val_loss: 3.8602 - val_accuracy:
```

Figure 11 : Progression de l'entraînement du modèle

3. Performances du modèle

Au bout de quelques itérations, le modèle est déjà précis (accuracy élevée) sur les données d'entraînement et de validations.

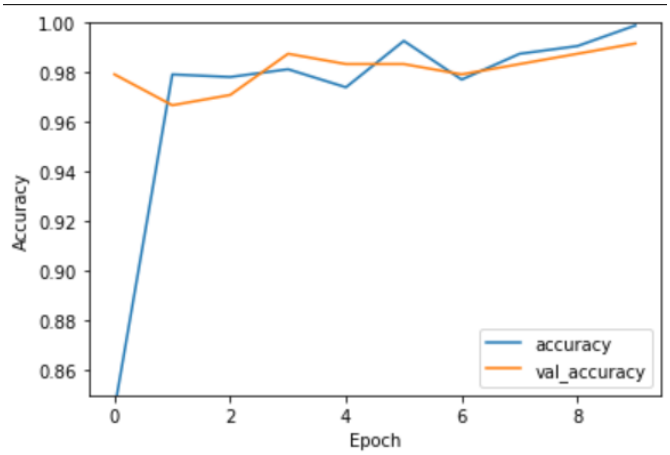

Figure 12 : Évaluation du modèle sur les données de test et de validation

On a les valeurs suivantes :

- loss (train) : `0.0043`
- accuracy (train) : `0.9990`
- val_loss (validation) : `2.3034`
- val_loss (validatio) : `0.9917`

La matrice de corrélation que l'on obtient corrobore ce qui a été dit précédemment.

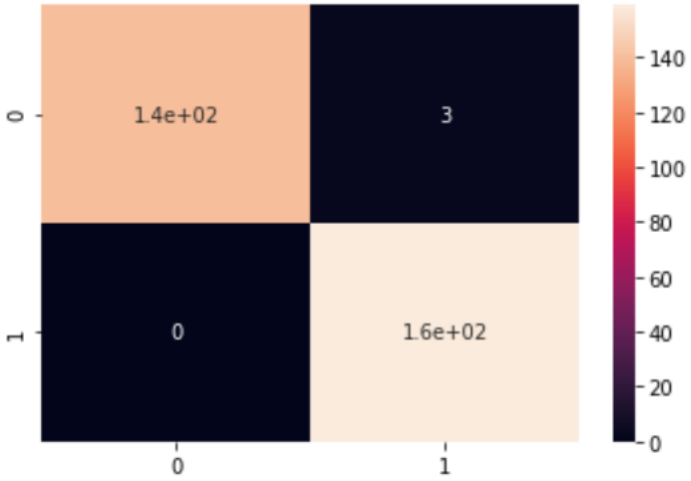

Figure 13 : Matrice de corrélation sur les données de test

Sur les données de test, seuls 3 images ont été mal répertoriées, où des masques ont été détectés là où il n'y en avait pas. On a donc une accuracy sur les données de test de : 0,990.

4. Application en temps réel

Notre but final est de pouvoir faire de la détection de masque en temps réel. Nous avons d'abord fait des essais sur des images fixes.

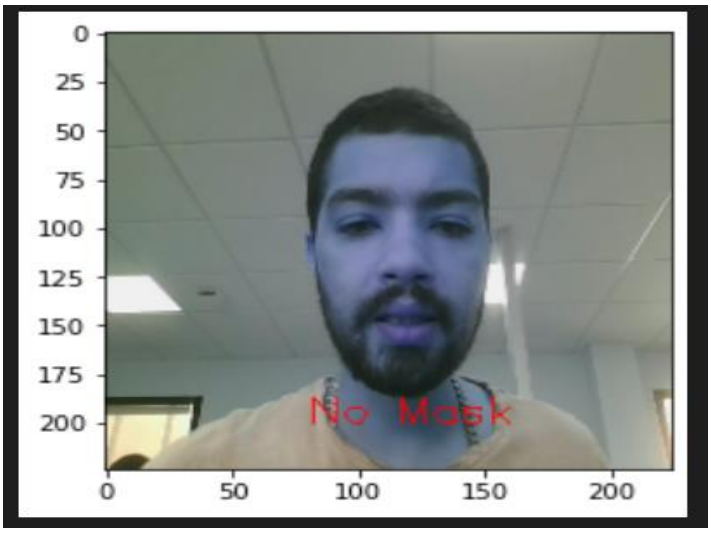

Figure 14 : Essai sur des images fixes

Nous configurons ensuite notre webcam et l'intégrons dans une boucle de programmation pour pouvoir traiter en continu les images du flux vidéo avec le programme développé précédemment.

```
cap = cv2.VideoCapture(0)
while True:
    # Reading the frame from the camera
    _,frame = cap.read()
    #Flipping the frame to see same side of yours
    frame = cv2.flip(frame, 1)
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    image_test=cv2.resize(frame, (224,224))

    image_test= np.expand_dims(image_test,axis=0)
    image_test.shape
    predictions = model.predict(image_test)
    label_predit=np.argmax(predictions)
    if label_predit==0 :
        message = "No Mask"
        cv2.putText(frame, message, (40, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
(0,0,255))
    else :
        message = "Masque"
        cv2.putText(frame, message, (40, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
(0,255,0))
    cv2.imshow("Mask Detector", frame)
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
# Release the camera and all resources
cap.release()
cv2.destroyAllWindows()
```

Figure 15 : Programme de détection de masque en temps-réel

Figure 16 : Application de détection de masque en temps-réel

5. Conclusion

Ce projet a été pour nous la découverte de l'approche du transfert learning, en plus d'une consolidation de ce qui a été vu précédemment sur les réseaux de neurones pour le traitement de l'image, en particulier avec Tensorflow et Keras.

Nous avons pu utiliser un modèle performant pour l'adapter à notre application. Les résultats sont encourageants et concrets, avec des performances élevées et un modèle assez robuste.

Nous avons utilisé une technique de data augmentation pour limiter l'overfitting et améliorer la généralisation du modèle

Le projet nous a donnée une réflexion sur les ressources informatiques et la problématique des longs temps de calculs, problème en parti pallié par l'approche de transfert learning et le calcul avec GPU.

Les applications semblent nombreuses et variées.