BRIEF SIMPLON - Détecteur de masque
Baptiste Le Berre & Thomas Chaigneau

Description des données

Le dataset est déjà séparé en quatre groupes de portraits en couleurs : jeu d'entraînement (avec et sans masque) et jeu de test (idem). Les images sont de tailles différentes, mais centrées sur les visages. Les images seront traitées par le code, déformées en arrays X de dimensions 100*100 en grayscale. L'array binaire de confirmation ou non de masque sera rajouté en y à partir du dossier d'emplacement. Pour terminer, les arrays X et y sont permutés suivant le même array aléatoire.

1. **Préparation des données**

- Import des différentes bibliothèques

```
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
```

- Transformation des images en matrices

```python
# Fonction permettant de récupérer les images dans le dossier
# Convertion en noir et blanc et redimensionnement en 100px/100px
def load_images_from_folder(folder):
    img = []
    for file in os.listdir(folder):
        i = cv2.imread(os.path.join(folder,file), cv2.IMREAD_GRAYSCALE)
        i = cv2.resize(i, (100,100))
        if i is not None:
            img.append(i)
    return img
```

```python
# Train avec masque
images = load_images_from_folder('train/with_mask')
Train_with_mask = np.array(images)
# Train sans masque
images = load_images_from_folder('train/without_mask')
Train_without_mask = np.array(images)
# Test avec masque
images = load_images_from_folder('test/with_mask')
Test_with_mask = np.array(images)
# Test sans masque
images = load_images_from_folder('test/without_mask')
Test_without_mask = np.array(images)
```

- Ajout des matrices des targets

```python
# Création des matrices target de 1 et 0
y_train_mask = np.ones((len(Train_with_mask), 1))
y_train_no_mask = np.zeros((len(Train_without_mask), 1))

# Création des matrices target de 1 et 0
y_test_mask = np.ones((len(Test_with_mask), 1))
y_test_no_mask = np.zeros((len(Test_without_mask), 1))
```

- Séparation en jeux d'entraînement et de test pour le modèle

```
# Jeu d'entraînement
y_train = np.concatenate((y_train_mask, y_train_no_mask))
X_train = np.concatenate((Train_with_mask, Train_without_mask))
# Jeu de test
y_test = np.concatenate((y_test_mask, y_test_no_mask))
X_test = np.concatenate((Test_with_mask, Test_without_mask))
```

- Mélange des différents jeux de données en vue de l'entraînement du modèle

```
# Mélange des jeux de données
array_random_train = np.random.permutation(len(y_train))
X_train = X_train[array_random_train]
y_train = y_train[array_random_train]
array_random_test = np.random.permutation(len(y_test))
X_test = X_test[array_random_test]
y_test = y_test[array_random_test]
```

2. Création du modèle

- Importation des bibliothèques

```
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt

from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import confusion_matrix
from img_to_matrix import X_train, X_test, y_train, y_test, load_images_from_folder
```

- Redimensionnement des images des jeux de données

```
# On redimensionne les images pour notre réseau de neurones
X_train = X_train.reshape(-1, 100, 100, 1)
X_test = X_test.reshape(-1, 100, 100, 1)
```

- Création des différentes couches du modèle

```
model = keras.Sequential()
model.add(layers.Conv2D(64, (3,3), activation='relu', input_shape=(100, 100, 1)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(128, (3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(256, (3,3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(2))
```

- On affiche le sommaire de nos couches pour vérifier les paramètres du modèle

```
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 98, 98, 64)        640
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 49, 49, 64)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 47, 47, 128)       73856
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 23, 23, 128)       0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 21, 21, 256)       295168
_________________________________________________________________
flatten_9 (Flatten)          (None, 112896)            0
_________________________________________________________________
dense_18 (Dense)             (None, 256)               28901632
_________________________________________________________________
dense_19 (Dense)             (None, 2)                 514
=================================================================
Total params: 29,271,810
Trainable params: 29,271,810
Non-trainable params: 0
_________________________________________________________________
```

3. Entraînement du modèle

- On compile notre modèle pour pouvoir l'entraîner

```
model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

- On entraîne notre modèle sur les données d'entraînement

```
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

```
Epoch 1/10
42/42 [==============================] - 18s 430ms/step - loss: 172.3273 - accuracy: 0.5768 - val_loss: 0.8134 - val_accuracy: 0.6598
Epoch 2/10
42/42 [==============================] - 19s 452ms/step - loss: 0.7167 - accuracy: 0.7277 - val_loss: 0.6982 - val_accuracy: 0.6443
Epoch 3/10
42/42 [==============================] - 19s 453ms/step - loss: 0.4525 - accuracy: 0.7922 - val_loss: 0.5199 - val_accuracy: 0.7526
Epoch 4/10
42/42 [==============================] - 19s 461ms/step - loss: 0.2808 - accuracy: 0.8926 - val_loss: 0.4868 - val_accuracy: 0.7680
Epoch 5/10
42/42 [==============================] - 19s 454ms/step - loss: 0.1885 - accuracy: 0.9265 - val_loss: 0.5442 - val_accuracy: 0.8041
Epoch 6/10
42/42 [==============================] - 19s 454ms/step - loss: 0.1918 - accuracy: 0.9238 - val_loss: 0.3958 - val_accuracy: 0.7990
Epoch 7/10
42/42 [==============================] - 19s 449ms/step - loss: 0.1458 - accuracy: 0.9448 - val_loss: 0.4355 - val_accuracy: 0.8454
Epoch 8/10
42/42 [==============================] - 19s 448ms/step - loss: 0.0697 - accuracy: 0.9773 - val_loss: 0.4969 - val_accuracy: 0.8299
Epoch 9/10
42/42 [==============================] - 19s 450ms/step - loss: 0.0532 - accuracy: 0.9809 - val_loss: 0.4873 - val_accuracy: 0.8814
Epoch 10/10
42/42 [==============================] - 19s 453ms/step - loss: 0.0171 - accuracy: 0.9939 - val_loss: 0.5445 - val_accuracy: 0.8711
```

4. Evaluation du modèle

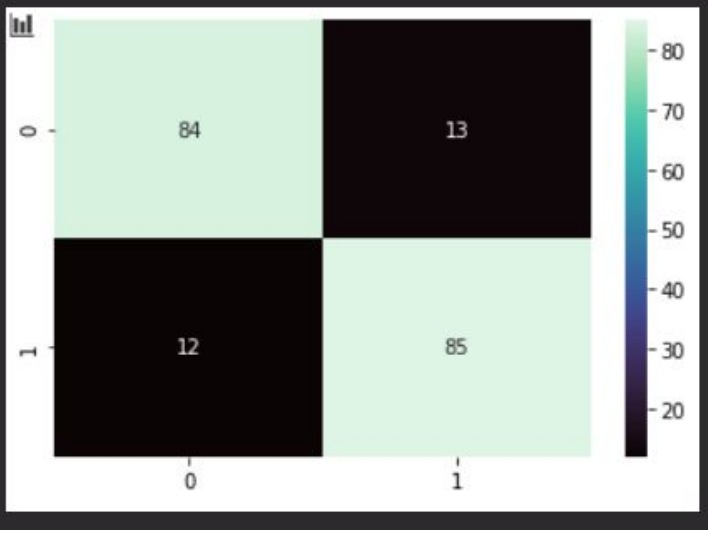
- On évalue notre modèle avant de l'utiliser en conditions réelles

```
[99]
pred = model.predict_classes(X_test)
```

```
[100]
conf = confusion_matrix(y_test, pred)
```

```
[101]
sns.heatmap(conf, cmap='mako', annot=True, fmt='d')
```

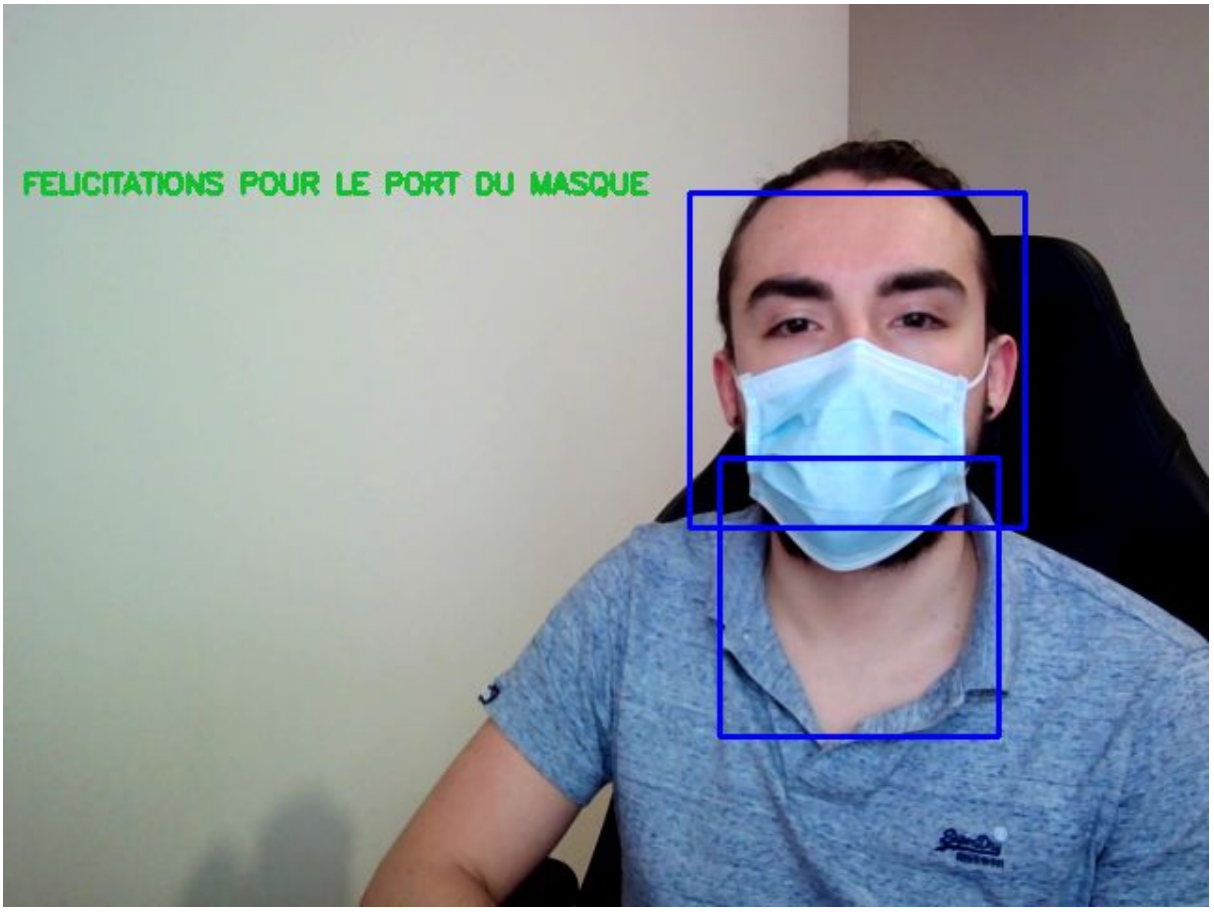
```
<AxesSubplot:>
```



- On sauvegarde le modèle pour pouvoir l'utiliser dans nos autres applications (photo et vidéo)

```
model.save('mask_model')
INFO:tensorflow:Assets written to: mask_model/assets
```

- Evaluation sur webcam :
- On récupère le feed de la webcam, puis on utilise Cascade de OpenCV pour identifier le visage
- On transforme les images individuelles au format adapté
- On teste le modèle sur les images créées
- On affiche un message adapté au résultat du test et une alerte son



Conclusion

Le modèle fonctionne plutôt bien même si la matrice de confusion présente quelques vrais négatifs et des faux positifs. Keras de Tensorflow semble donc bien adapté pour ce modèle et son application sur les photos et des flux vidéos.
Le modèle reste efficace malgré la transformation en grayscale 100*100px.

Inconvénients :

- Le modèle demande une reconnaissance préalable du visage et fonctionne donc surtout de face.
- On a peut-être un peu gonflé le nombre de paramètres du modèle en utilisant des couches très grandes. Nous aurions éventuellement obtenu la même chose avec un modèle moins lourd.