Rapport projet 13 : Modèle intelligent pour la détection des masques

1- Préparation des données

Nous commençons par importer nos données qui sont stockées dans deux dossiers. Le premier contient les images avec masque et le deuxième contient les images sans masque.

```
directory = "/content/drive/MyDrive/Mask_Data/with_mask/"

X = []
# liste pour enregistrer les labels
y = []
# l va servir à numéroter les labels : dans la boucle, l s'incrémente de 1 à chaque passage
l = 1

for x in tqdm(os.listdir(directory)):
    image = cv2.imread(directory + x)
    # on resize
    image = cv2.resize(image,(224,224), interpolation = cv2.INTER_AREA)
    X.append(image)
    y.append(l)
```

```
100%|██████████| 755/755 [00:13<00:00, 54.93it/s]
```

```
directory = "/content/drive/MyDrive/Mask_Data/without_mask/"


l= 0

for x in tqdm(os.listdir(directory)):
    image = cv2.imread(directory + x)
    # on resize
    image = cv2.resize(image,(224,224), interpolation = cv2.INTER_AREA)
    X.append(image)
    y.append(l)
```

```
100%|██████████| 753/753 [00:12<00:00, 62.14it/s]
```

Nous stockons les images dans la variable X tandis que y contient les labels : 0 pour les photos sans masque et 1 pour les photos avec masque. La labellisation des images se fait dans la boucle for, tout simplement selon le dossier qui est importé. On en profite pour resize les images (224, 224).

Enfin, on transforme les variables X et y en tableau numpy tout en divisant la variable X par 255. De plus, on catégorise la variable y avec la méthode to_categorical de keras.

2- Séparation du jeu de données

On effectue une train_test_split (sklearn) sur X et y pour obtenir notre jeu d'entraînement et de test. On en profite pour créer un jeu de validation :

```
# Création du jeu de validation
X_train_val, X_val, y_train_val, y_val = train_test_split(X_train, y_train, test_size=0.10, random_state=1)
X_train_val.shape, X_val.shape, y_train_val.shape, y_val.shape

X_train_val = np.array(X_train_val)
y_train_val = np.array(y_train_val)
```

On augmente également le nombre de nos images grâce à la datatransformation qui consiste à augmenter notre jeu de données en modifiant légèrement les images existantes (inclinaison, luminosité, contraste, sens de l'image…). On utilise pour cela l'outil ImageDataGenerator de keras.

```
## Data transformation

from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import ImageDataGenerator

train_data_generator = ImageDataGenerator(

        # data augmentation
        rotation_range = 10,
        width_shift_range = 0.1,
        height_shift_range = 0.1,
        zoom_range = 1.1,
        horizontal_flip = True
        )

data_gen = train_data_generator.flow(X_train, y_train, batch_size = 64)
```

3- Architecture CNN sur Tensorflow

On utilise la technologie Vgg16 : grâce au transfer learning, on obtient le réseau de neurones et il ne nous reste qu'à personnaliser les trois dernières couches du réseau.

```
# Modèle VGG16
# base_model = VGG16(weights='imagenet', include_top=True)
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
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
58900480/58889256 [==============================] - 1s 0us/step
```

```
num_classes = 2

model = tf.keras.Sequential([
    base_model,
    Flatten(),
    Dense(4096,activation='relu'),
    Dense(4096, activation='relu'),
    Dense(num_classes, activation='softmax')
    ])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 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
```

Compilation du modèle :

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

Entraînement du modèle :

```
from keras.callbacks import ModelCheckpoint, EarlyStopping

checkpoint = ModelCheckpoint("vgg16_1.h5", monitor='val_accuracy', verbose=1, save_best_only=True, save_weights_only=False, mode='auto', period=1)
# early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1, mode='auto')

history = model.fit(data_gen,
                    epochs=5,
                    validation_data = (X_train_val, y_train_val),
                    callbacks=[checkpoint])
```

```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify the frequency in number of batches seen.
Epoch 1/5
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
19/19 [==============================] - ETA: 0s - loss: 9.6794 - accuracy: 0.4760
Epoch 1: val_accuracy improved from -inf to 0.55760, saving model to vgg16_1.h5
19/19 [==============================] - 49s 2s/step - loss: 9.6794 - accuracy: 0.4760 - val_loss: 0.5493 - val_accuracy: 0.5576
Epoch 2/5
19/19 [==============================] - ETA: 0s - loss: 0.5476 - accuracy: 0.7164
Epoch 2: val_accuracy improved from 0.55760 to 0.97419, saving model to vgg16_1.h5
19/19 [==============================] - 28s 1s/step - loss: 0.5476 - accuracy: 0.7164 - val_loss: 0.1349 - val_accuracy: 0.9742
Epoch 3/5
19/19 [==============================] - ETA: 0s - loss: 0.3635 - accuracy: 0.8234
Epoch 3: val_accuracy did not improve from 0.97419
19/19 [==============================] - 19s 1s/step - loss: 0.3635 - accuracy: 0.8234 - val_loss: 0.1295 - val_accuracy: 0.9401
Epoch 4/5
19/19 [==============================] - ETA: 0s - loss: 0.2668 - accuracy: 0.8905
Epoch 4: val_accuracy improved from 0.97419 to 0.98065, saving model to vgg16_1.h5
19/19 [==============================] - 26s 1s/step - loss: 0.2668 - accuracy: 0.8905 - val_loss: 0.0470 - val_accuracy: 0.9806
Epoch 5/5
19/19 [==============================] - ETA: 0s - loss: 0.2099 - accuracy: 0.9080
Epoch 5: val_accuracy improved from 0.98065 to 0.98710, saving model to vgg16_1.h5
19/19 [==============================] - 26s 1s/step - loss: 0.2099 - accuracy: 0.9080 - val_loss: 0.0273 - val_accuracy: 0.9871
```