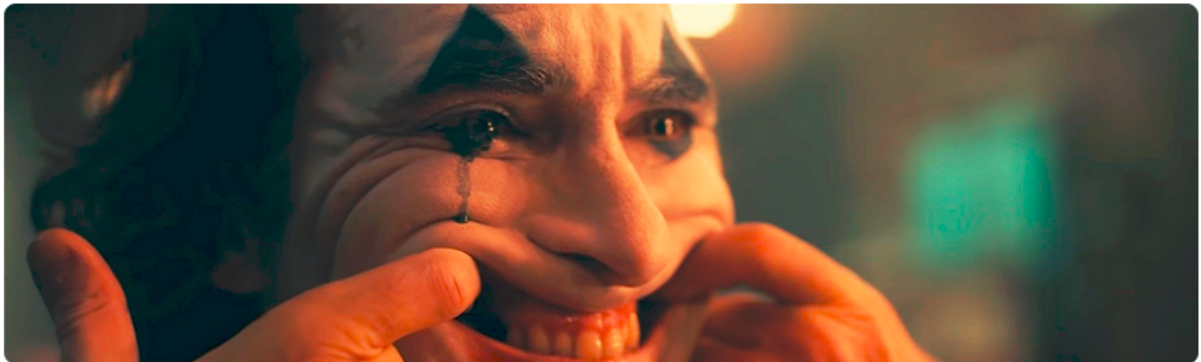


Détecteur des émotions par les expressions faciales

Jérémy et Aude



Contexte du projet

Les expressions du visage peuvent naturellement servir à évaluer la satisfaction d'un client aux prises avec un service après-vente ou à face à un produit récemment acquis dont il s'agit de comprendre le fonctionnement. On peut encore mentionner les applications suivantes :

- La détection d'un manque d'attention chez un conducteur en vue d'augmenter la sécurité de la conduite.
- L'évaluation du niveau de stress de passagers à l'atterrissage ou à l'arrivée en gare ou la détection de comportements suspects.
- L'humanisation des robots dans leurs interactions avec les humains dont ils prendraient en compte l'état psychique.

Ce projet a pour but, à partir d'une base de données d'images représentant sept émotions (joie, surprise, tristesse, colère, dégoût, peur, neutralité), d'implémenter un modèle permettant de détecter cette émotion sur un flux vidéo et d'y associer un emoji correspondant.

Ce projet nous a permis de nous familiariser avec **Google Colab**.

```
1 #mount google drive
2 from google.colab import drive
3 drive.mount('/content/gdrive', force_remount=True)
```

Nous avons d'abord uploadé le dataset sur notre google drive, en version zippée, et l'avons dézippée sur le drive.

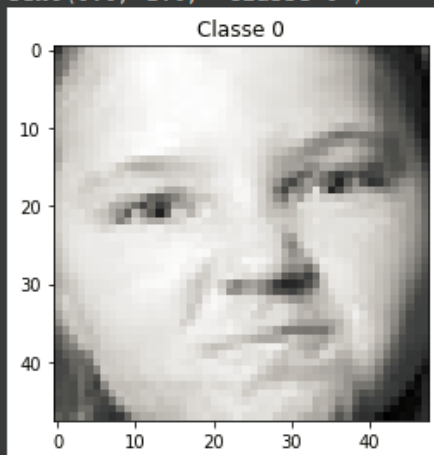
```
1 #unzip file in the emotions folder
2 #!unzip '/content/gdrive/MyDrive/emotions/data_emotion.zip' -d "/content/gdrive/MyDrive/emotions/"
```

Cette façon de faire est beaucoup plus rapide que d'utiliser l'**outil Unzipper** de Google. Puis nous avons utilisé l'outil **glob.glob** et **cv2.imread** pour récupérer les vecteurs représentant les images. Cette opération a été assez lente (environ cinq heures). Nous avons depuis découvert qu'il y a un outil de **tensorflow** qui permet de le réaliser plus rapidement, en tirant profit du GPU mis à disposition sur google colab.

Puis nous avons créé la variable target à partir du nom du fichier dans lequel se trouvait l'image.

```
1 #X_train = np.array(X_train).reshape(-1, IMG_SIZE, IMG_SIZE)
2 plt.imshow(X_train[600], cmap='gray', interpolation='none')
3 plt.title("Classe {}".format(y_train[600]))
```

```
Text(0.5, 1.0, 'Classe 0')
```



Enfin, nous avons reshapé le X_train et utilisé **label binarizer** afin de convertir la valeur de y en un vecteur binaire:

```
1 from sklearn.preprocessing import LabelBinarizer
2 lb = LabelBinarizer()
3 y_train = lb.fit_transform(y_train)
4 y_test = lb.fit_transform(y_test)
```

Puis nous avons pu commencer l'entraînement.

```
1 model = Sequential()
2
3 model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48,48,1)))
4 model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
5 model.add(MaxPooling2D(pool_size=(2, 2)))
6 model.add(Dropout(0.25))
7
8 model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
9 model.add(MaxPooling2D(pool_size=(2, 2)))
10 model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
11 model.add(MaxPooling2D(pool_size=(2, 2)))
12 model.add(Dropout(0.25))
13
14 model.add(Flatten())
15 model.add(Dense(1024, activation='relu'))
16 model.add(Dropout(0.5))
17 model.add(Dense(7, activation='softmax'))
18
19 model.summary()
```

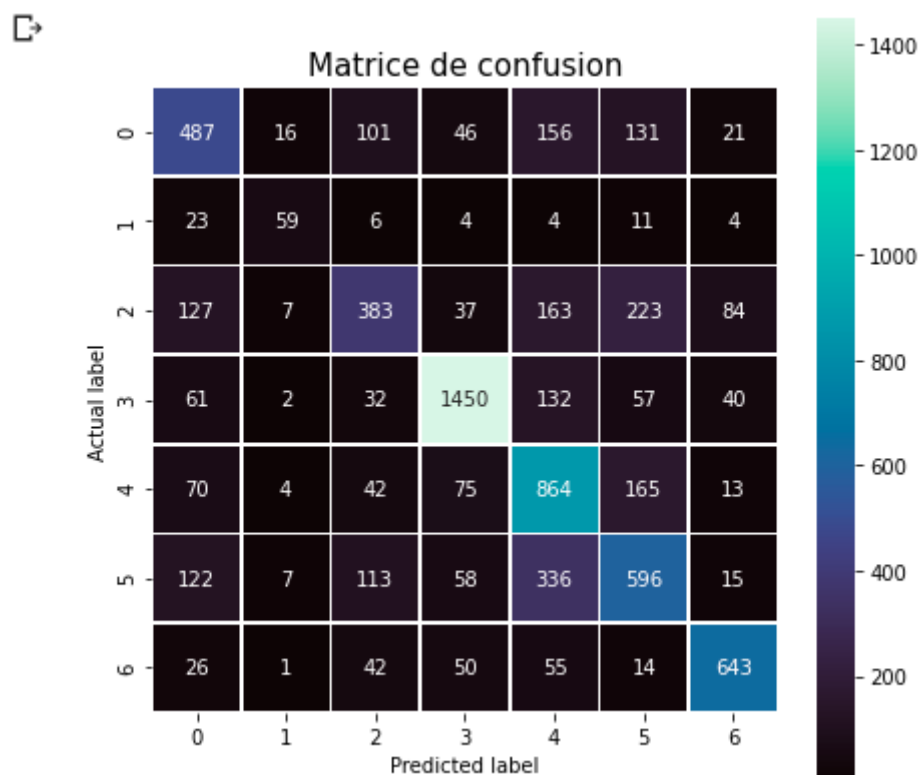
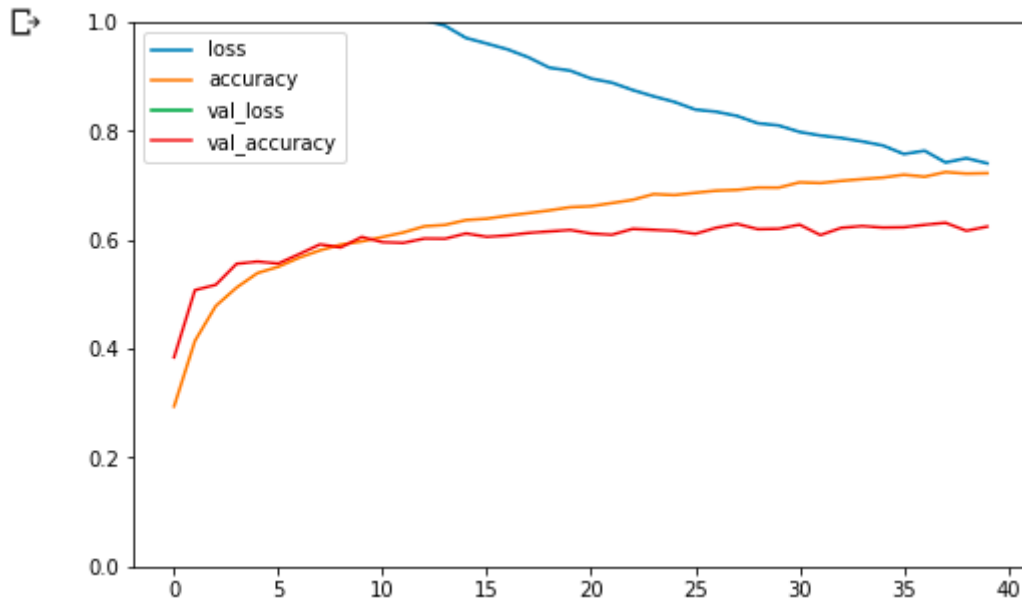
Après un petit souci qui venait à priori d'un bug de colab (une réinitialisation de l'environnement a suffit):

```
... Epoch 1/100
898/898 [=====] - 7s 7ms/step - loss: 1.8113 - accuracy: 0.2499 - val_loss: 1.8131 - val_accuracy: 0.2471
Epoch 2/100
898/898 [=====] - 6s 7ms/step - loss: 1.8091 - accuracy: 0.2514 - val_loss: 1.8131 - val_accuracy: 0.2471
Epoch 3/100
898/898 [=====] - 6s 7ms/step - loss: 1.8095 - accuracy: 0.2473 - val_loss: 1.8133 - val_accuracy: 0.2471
Epoch 4/100
898/898 [=====] - 6s 6ms/step - loss: 1.8077 - accuracy: 0.2504 - val_loss: 1.8131 - val_accuracy: 0.2471
Epoch 5/100
898/898 [=====] - 6s 6ms/step - loss: 1.8087 - accuracy: 0.2553 - val_loss: 1.8132 - val_accuracy: 0.2471
Epoch 6/100
898/898 [=====] - 6s 6ms/step - loss: 1.8109 - accuracy: 0.2536 - val_loss: 1.8131 - val_accuracy: 0.2471
Epoch 7/100
898/898 [=====] - 6s 6ms/step - loss: 1.8115 - accuracy: 0.2506 - val_loss: 1.8131 - val_accuracy: 0.2471
Epoch 8/100
898/898 [=====] - 6s 7ms/step - loss: 1.8075 - accuracy: 0.2541 - val_loss: 1.8131 - val_accuracy: 0.2471
Epoch 9/100
898/898 [=====] - 6s 7ms/step - loss: 1.8097 - accuracy: 0.2515 - val_loss: 1.8132 - val_accuracy: 0.2471
Epoch 10/100
898/898 [=====] - 6s 7ms/step - loss: 1.8077 - accuracy: 0.2538 - val_loss: 1.8131 - val_accuracy: 0.2471
```

et finalement nous avons obtenu 63% d'accuracy sur les données de validation:

```
Epoch 98/100
898/898 [=====] - 6s 7ms/step - loss: 0.1104 - accuracy: 0.9609 - val_loss: 1.6389 - val_accuracy: 0.6344
Epoch 99/100
898/898 [=====] - 6s 7ms/step - loss: 0.1160 - accuracy: 0.9591 - val_loss: 1.6521 - val_accuracy: 0.6379
Epoch 100/100
898/898 [=====] - 6s 7ms/step - loss: 0.1128 - accuracy: 0.9612 - val_loss: 1.6626 - val_accuracy: 0.6329
```

L'entraînement aurait pu durer moins longtemps, 40 époques auraient sans doute donné le même résultat. Après de nombreux essais de différents paramètres, nous **n'avons pas réussi à obtenir un score plus élevé** et avons décidé d'utiliser le modèle tel quel.



Puis pour la version webcam, nous avons utilisé **haarcascade** et **cv2**. Nous avons créé un ROI des emojis pour pouvoir les insérer dans la vidéo.



Enfin, nous avons ajouté un son pour poser une question sur les émotions des gens.

Conclusions

Le modèle reconnaît assez moyennement les émotions, il faut vraiment forcer les traits pour qu'une émotion soit identifiée. Cela est dû en partie au faible taux d'accuracy du modèle (63%), mais aussi au dataset de base, dont les photos ne sont pas extraordinaires. De plus, le dataset est extrêmement déséquilibré, avec par exemple environ 400 images pour disgust et plus de 7000 pour happy. Cela crée un déséquilibre.

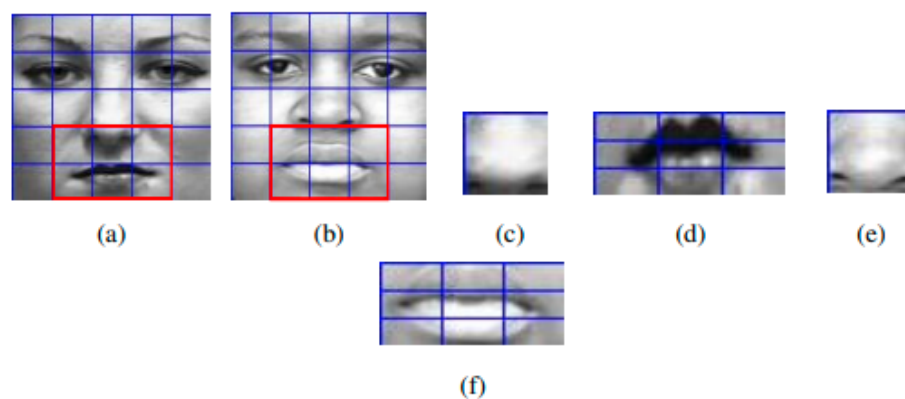
Pour palier à ces problèmes, plusieurs solutions nous sont présentées :

Un modèle déjà existant aurait pu être utilisé :

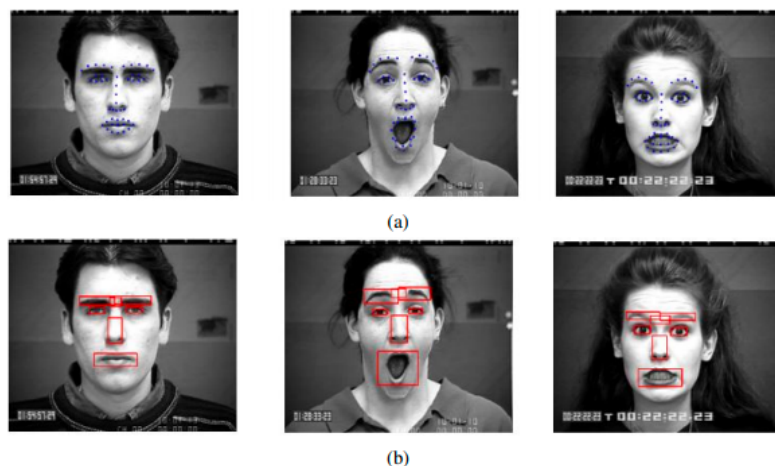
```
In [7]: predictions = DeepFace.analyze(img)
Action: race: 100% | 4/4 [00:00<00:00, 4.35it/s]
```

DeepFace est un package léger de reconnaissance faciale et d'analyse des attributs faciaux pour Python. Il prend en charge les modèles de reconnaissance faciale les plus populaires, notamment VGG-Face, Google FaceNet, OpenFace, Facebook DeepFace. En outre, il peut analyser les attributs faciaux tels que les prédictions d'émotion, d'âge, de sexe et de race dans son module d'analyse des attributs faciaux. Ce package utilisable rapidement donne des résultats satisfaisants.

Pour un modèle plus poussé spécifiquement sur la reconnaissance des émotions, plusieurs axes sont abordés dans les ouvrages de recherches comme la thèse de *Khadija Lekdioui* (Reconnaissance d'états émotionnels par analyse visuelle du visage et apprentissage machine, 2019). L'une de ces problématiques est la détection des émotions du visage sous différents angles (de profil par exemple). Une analyse spécifique des différentes régions du visage pourrait être réalisée pour accroître l'efficacité d'un modèle de reconnaissance.



(a) Le visage entier en tant que ROI, la région rouge incorpore deux composants faciaux qui sont la bouche et une partie du nez. (b) Le visage entier en tant que ROI, la région rouge incorpore seulement un composant facial qui est la bouche. (c) et (d) les ROIs nez et bouche extraites de (a). (e) et (f) les ROIs nez et bouche extraites de (b). D'après Khadija Lekdioui, 2019.



(a) Exemples de 49 points caractéristiques détectés par SDM*. (b) Exemples d'extraction de ROIs D'après Khadija Lekdioui, 2019.

SDM : La méthode de descente supervisée (SDM) est l'une des principales approches de régression en cascade pour l'alignement du visage avec des performances de pointe et une base théorique solide.