Détecteur des émotions par les expressions faciales

Jérémy et Aude

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.

```
#mount google drive
from google.colab import drive
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.

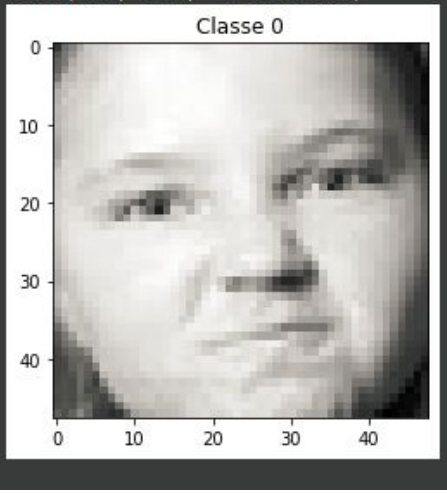
```
#unzip file in the emotions folder
#!unzip '/content/gdrive/MyDrive/emotions/data_emotion.zip' -d "/content/gdrive/MyDrive/emotions/"
```

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.

```
#X_train = np.array(X_train).reshape(-1, IMG_SIZE, IMG_SIZE)
plt.imshow(X_train[600], cmap='gray', interpolation='none')
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

```
from sklearn.preprocessing import LabelBinarizer
lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_test = lb.fit_transform(y_test)
```

Puis nous avons pu commencer l'entraînement.

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48,48,1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))

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.