La reconnaissance des expressions faciales

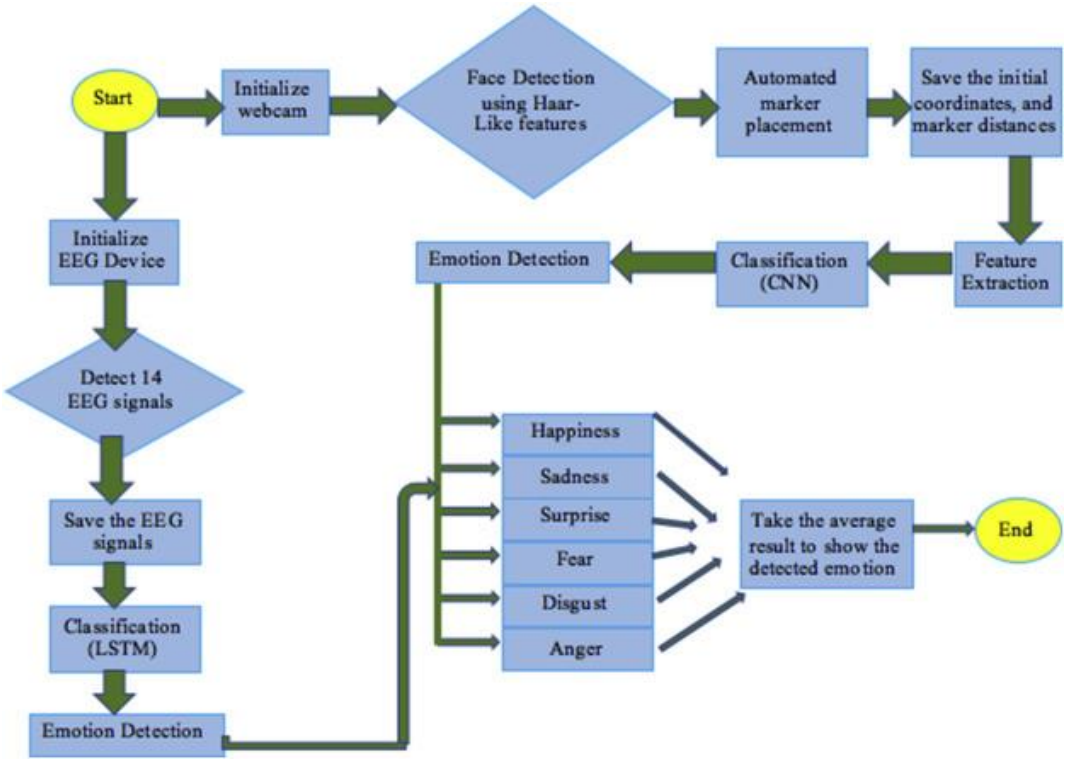
JUIN 2023

Faculté des Sciences
كلية العلوم
Université Ibn Tofail
جامعة ابن طفيل

Encadré par : Adnani Younes
Rédigé par : EL-asri Soufia

Introduction Générale

L'expression faciale est l'une des principales façons dont les humains affichent leurs émotions. La reconnaissance des expressions faciales est l'un des moyens les plus puissants, naturels et immédiats pour les êtres humains de communiquer leurs émotions et leurs intentions. Les humains peuvent être dans certaines circonstances empêchés de montrer leurs émotions, comme les patients hospitalisés, ou en raison de carences ; par conséquent, une meilleure reconnaissance des autres émotions humaines conduira à une communication efficace. La reconnaissance automatique des émotions humaines a reçu beaucoup d'attention récemment avec l'introduction de l'IOT et des environnements intelligents dans les hôpitaux, les maisons intelligentes et les villes intelligentes. Les assistants personnels intelligents (IPA), tels que Siri, Alexia, Cortana et d'autres, utilisent le traitement du langage naturel pour communiquer avec les humains, mais lorsqu'il est augmenté d'émotions, il augmente le niveau de communication efficace et d'intelligence au niveau humain.



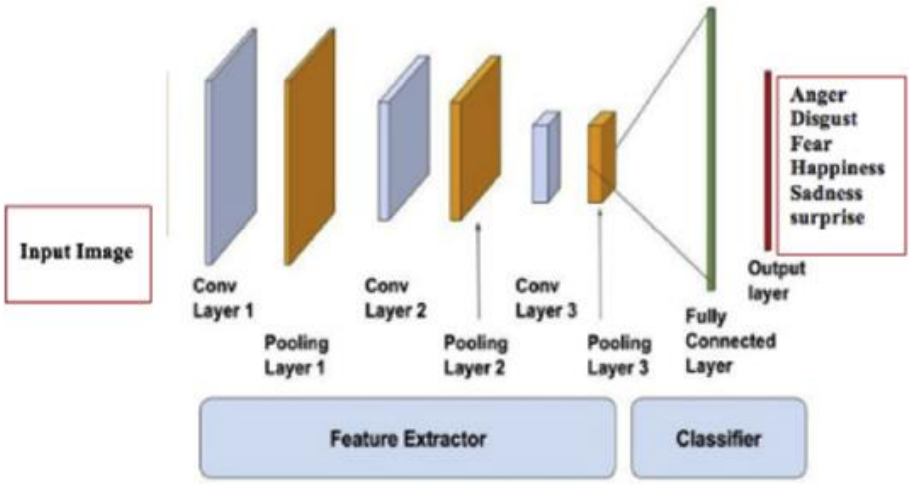
L’objectif :

Les expressions faciales sont une forme de communication non verbale importante dans les interactions humaines. La reconnaissance des expressions faciales à partir d'images peut être utilisée dans divers domaines, tels que la psychologie, l'ergonomie, les interfaces homme-machine, la réalité virtuelle, la surveillance de la sécurité, etc.

L'objectif de notre projet, est de développer un système capable d'analyser et d'interpréter les expressions faciales humaines. Il s'agit d'une application de la vision par ordinateur et de l'apprentissage automatique qui vise à comprendre les émotions et les états émotionnels des individus à partir de leurs visages.

Deux réseaux neuronaux convolutifs ont été utilisé dans notre système pour obtenir une détection améliorée des émotions faciales telle qu'elle est appliquée à d'autres domaines informatiques tels que la reconnaissance faciale et la détection d'objets. De plus, les prédictions sont basées sur des informations données à un moment donné.

La figure montre la structure du réseau utilisée pour la détection des émotions à l'aide de repères faciaux. Ce réseau prend une image d'entrée et tente de prédire l'émotion de sortie.



L’étude préliminaire :

Revue de littérature : Dans cette étape, nous effectuons une recherche approfondie des travaux existants dans le domaine de la reconnaissance des expressions faciales. Nous examinons des articles scientifiques, tel que :

- **"Facial expression recognition : A Survey" par Ms. YUNXIN HUANG, Dr. Fei Chen, Shaohe Lv, et Prof. Xiaodong Wang. (2019) :** Dans cette enquête, les méthodes de FER existantes sont d'abord classées en deux groupes principaux : les approches conventionnelles et les approches basées sur l'apprentissage profond. Pour mettre en évidence les différences et les similitudes, nous proposons une structure générale d'approche conventionnelle de la FER et passons en revue les technologies possibles qui peuvent être utilisées dans chaque composant.
- **“Emotion recognition in the wild from videos using images” par Sarah Bargal, Emad Barsoum, Cristian Canton Ferrer, Cha Zhang (2016) :** Cet article présente les détails de mise en œuvre de la solution proposée pour le défi Emotion Recognition in the Wild 2016, dans la catégorie de la reconnaissance des émotions basée sur la vidéo.
- **“Emotion recognition in doctor-patient interactions from real-world clinical video database : Initial development of artificial empathy” par Chih-Wei Huang, Bethany C.Y. Wu, Phung Anh Nguyen, Hsiao-Han Wang, Chih-Chung Kao, Pei-Chen Lee, Annisa Ristya Rahmanti, Jason C. Hsu, Hsuan-Chia Yang, Yu-Chuan Jack Li (2023) :** L'utilisation prometteuse de l'intelligence artificielle (IA) pour émuler l'empathie humaine peut aider un médecin à s'engager dans une relation médecin-patient plus empathique. Cette étude démontre l'application de l'empathie artificielle basée sur la reconnaissance des émotions faciales pour évaluer les relations médecin-patient dans la pratique clinique.

pour comprendre les avancées récentes, les méthodes utilisées, les défis rencontrés et les lacunes potentielles dans la littérature existante.

Collecte de données : Nous identifions les bases de données disponibles contenant des images d'expressions faciales, telles que le Facial Expression Recognition and Analysis (FERA) dataset, le CK+ dataset, le Facial Expression Recognition de kaggle-dataset ou l'Extended Cohn-Kanade (CK+) dataset. Nous examinons également les conditions d'acquisition des données, telles que l'éclairage, les angles de vue et la diversité des expressions couvertes.

Évaluation de la faisabilité : Nous évaluons la faisabilité du projet en termes de ressources nécessaires, telles que le temps, le matériel informatique, les compétences en programmation et en apprentissage automatique, ainsi que l'accès aux bases de données pertinentes. Nous identifions également les contraintes potentielles, telles que les limites de la précision de la reconnaissance des expressions faciales et les défis de la généralisation du modèle à des visages inconnus.

Objectifs de recherche : Nous définissons les objectifs spécifiques de notre recherche. Par exemple, notre objectif principal peut être de développer un modèle de reconnaissance des expressions faciales précis et robuste, capable de reconnaître les expressions émotionnelles de base telles que la joie, la tristesse, la colère, la peur, le dégoût et la surprise.

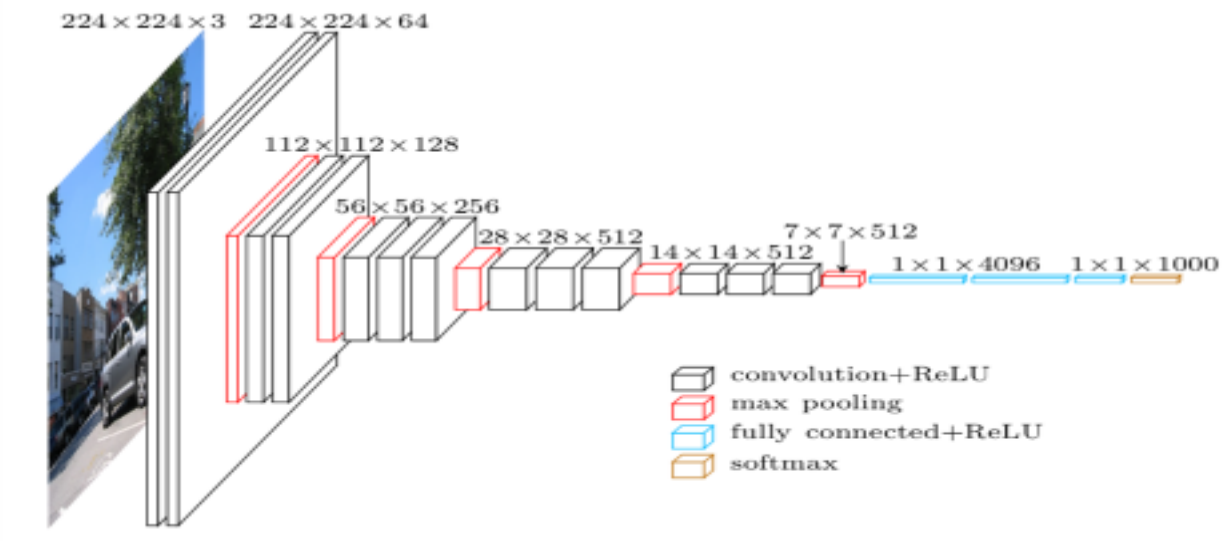
Méthodes et approches possibles : Nous examinons différentes approches possibles pour la reconnaissance des expressions faciales, telles que l'utilisation de réseaux de neurones convolutifs (CNN), de descripteurs de texture ou de techniques de traitement d'images. Nous évaluons également les avantages et les inconvénients de chaque approche en fonction de nos objectifs et des ressources disponibles.

Planification de la recherche : Nous élaborons une planification préliminaire du projet, y compris la collecte des données supplémentaires si nécessaire, la mise en œuvre des algorithmes de reconnaissance des expressions faciales, l'évaluation des performances du modèle, l'analyse des résultats et la rédaction du rapport final.

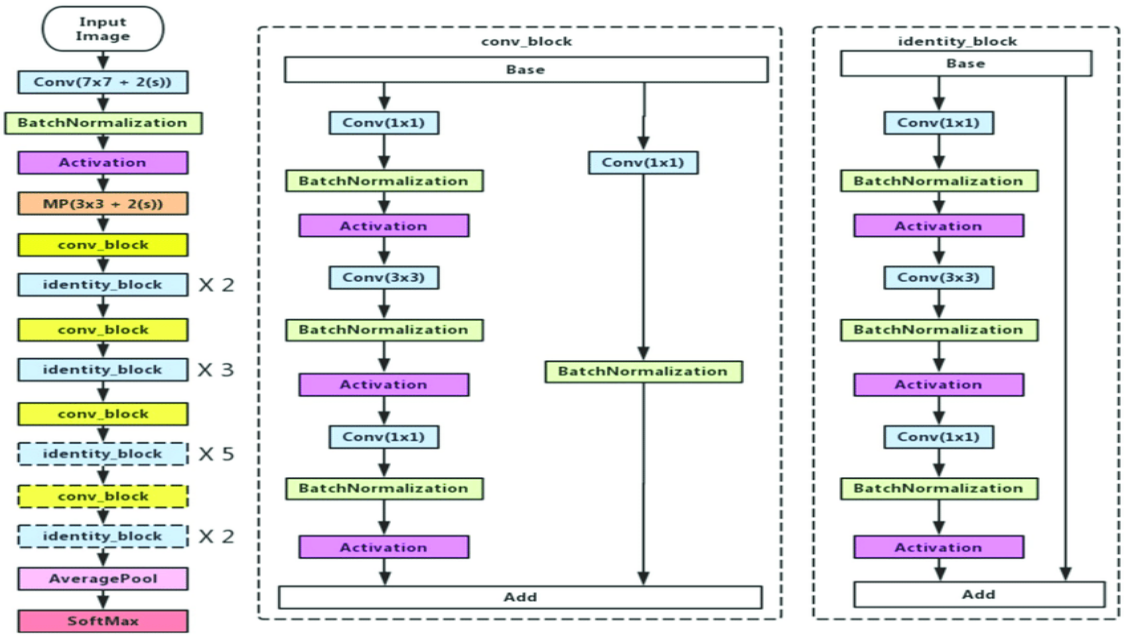
Les approches utilisées :

Pour la construction de notre système, nous utilisons deux approches basées sur des réseaux de neurones convolutifs : VGG16 et ResNet50. Ces deux architectures sont largement utilisées dans le domaine de la vision par ordinateur et ont démontré de bonnes performances dans la reconnaissance des expressions faciales.

VGG16 : est une architecture de réseau neuronal à convolution (CNN) qui a été utilisée pour remporter le concours ILSVR (Imagenet) en 2014. Elle est considérée comme l'une des meilleures architectures de modèle de vision jusqu'à ce jour. La chose la plus unique à propos de VGG16 est qu'au lieu d'avoir un grand nombre d'hyper-paramètres, ils se sont concentrés sur des couches de convolution de filtre 3x3 avec une foulée 1 et ont toujours utilisé le même rembourrage et la même couche maxpool de filtre 2x2 de foulée 2. Il suit cet arrangement de couches de convolution et de pool maximum de manière cohérente dans toute l'architecture. Au final, il a 2 FC (couches entièrement connectées) suivies d'un softmax pour la sortie. Le 16 dans VGG16 fait référence à 16 couches qui ont des poids. Ce réseau est un assez grand réseau et il compte environ 138 millions (environ) de paramètres.



ResNet50 : est une architecture de réseau de neurones convolutifs profonde qui a été introduite dans l'article "Deep Residual Learning for Image Recognition" par K. He, X. Zhang, S. Ren, et J. Sun en 2015. Ce réseau est une extension du modèle ResNet original, qui a été conçu pour résoudre le problème de dégradation de la performance lors de l'augmentation de la profondeur du réseau.
La particularité de ResNet50 réside dans l'utilisation de blocs résiduels, également appelés "skip connections" ou connexions sautées. Ces connexions permettent aux informations de contourner certains blocs de convolution, ce qui facilite l'apprentissage en profondeur et aide à prévenir la diminution de la performance.
ResNet50 se compose de 50 couches et utilise des blocs résiduels pour faciliter l'apprentissage en profondeur. Chaque bloc résiduel se compose de plusieurs couches de convolution, de normalisation de lot et d'activation, ainsi que d'une connexion sautée qui ajoute l'entrée d'un bloc à sa sortie. Cette architecture permet d'apprendre des représentations de haute qualité à partir des données d'entrée et a démontré d'excellentes performances dans diverses tâches de vision par ordinateur, y compris la reconnaissance des expressions faciales.
En utilisant ResNet50, nous pouvons bénéficier de sa capacité à extraire des caractéristiques discriminantes à partir des images d'expressions faciales. Ces caractéristiques peuvent ensuite être utilisées pour entraîner un modèle de reconnaissance des expressions faciales précis et robuste.



En utilisant VGG16 et ResNet50 comme bases, nous pouvons exploiter les capacités d'apprentissage de ces réseaux pour extraire des caractéristiques significatives à partir des images d'expressions faciales. Ces caractéristiques seront ensuite utilisées pour entraîner notre modèle de reconnaissance des expressions faciales, améliorant ainsi la précision et la robustesse du système.

En utilisant deux approches différentes, nous pouvons également comparer leurs performances et évaluer laquelle est la plus adaptée à notre tâche spécifique de reconnaissance des expressions faciales.

Script :

Les méthodes utilisées :

Importation des bibliothèques:

```
[ ] import numpy as np
    import pandas as pd
    import seaborn as sn
    import matplotlib.pyplot as plt
    import skimage.io
    from tqdm import tqdm
    import cv2
    import os
    import keras.backend as K
    import tensorflow as tf
    import tensorflow_hub as hub
    from sklearn.preprocessing import LabelBinarizer
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score, confusion_matrix
    import itertools
    import plotly.graph_objs as go
    from plotly.offline import init_notebook_mode, iplot
    from plotly import tools
    from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D,BatchNormalization,Activation
    from keras.preprocessing.image import ImageDataGenerator
    from keras.applications.vgg16 import VGG16, preprocess_input
    from tensorflow.keras.applications import ResNet50
    from keras.applications.nasnet import NASNetLarge
    from keras import layers
    from keras.models import Model, Sequential
    from keras.optimizers import Adam, RMSprop
    from keras.callbacks import  ReduceLROnPlateau, ModelCheckpoint,EarlyStopping
    from keras.preprocessing.image import ImageDataGenerator
    init_notebook_mode(connected=True)
    RANDOM_SEED = 123
```

Charger les données et tracer les échantillons :

Ce code extrait tous les fichiers et dossiers contenus dans une archive ZIP nommée "data_DL.zip" dans le répertoire actuel où se trouve le script Python. Il utilise la bibliothèque zipfile de Python pour effectuer cette opération. Une fois que l'extraction est terminée, il affiche le message "Done".

```
[ ] from zipfile import ZipFile
    file_name = "data_DL.zip"
    with ZipFile(file_name,'r') as zip:
      zip.extractall()
      print('Done')
```

```
Done
```

Cette fonction charge les images d'un répertoire, les redimensionne et les normalise, puis renvoie les tableaux d'images, les étiquettes correspondantes et un dictionnaire de correspondance entre les identifiants de classe et les noms de classe.

```
[ ]  TRAIN_DIR = ('Training/Training/')
     TEST_DIR = ('Testing/Testing/')
```

```
[ ]  def load_data(dir_path, IMG_SIZE):

        X = []
        y = []
        i = 0
        labels = dict()
        for path in tqdm(sorted(os.listdir(dir_path))):
            if not path.startswith('.'):
                labels[i] = path
                for file in os.listdir(dir_path + path):
                    if not file.startswith('.'):
                        img = cv2.imread(dir_path + path + '/' + file)
                        img = img.astype('float32') / 255
                        resized = cv2.resize(img, IMG_SIZE, interpolation = cv2.INTER_AREA)
                        X.append(resized)
                        y.append(i)
                i += 1
        X = np.array(X)
        y = np.array(y)
        print(f'{len(X)} images chargées à partir le répertoire: {dir_path}.')
        return X, y, labels
```

Cette fonction permet de visualiser un échantillon d'images à partir des données fournies, en les affichant dans une grille avec un titre correspondant à la classe de chaque échantillon. Cela peut être utile pour examiner les données et vérifier si elles sont correctement chargées et préparées avant de les utiliser pour l'entraînement du modèle.

```
[ ]  def plot_samples(X, y, labels_dict, n=50):

         for index in range(len(labels_dict)):
             imgs = X[np.argwhere(y == index)][:n]
             j = 10
             i = int(n/j)

             plt.figure(figsize=(10,3))
             c = 1
             for img in imgs:
                 plt.subplot(i,j,c)
                 plt.imshow(img[0])

                 plt.xticks([])
                 plt.yticks([])
                 c += 1
             plt.suptitle(labels_dict[index])
             plt.show()
```

Classes d'encodage :

En utilisant la méthode to_categorical(), un tableau numpy (ou) un vecteur qui a des nombres entiers qui représentent différentes catégories, peut être converti en un tableau numpy (ou) une matrice qui a des valeurs binaires et a des colonnes égales au nombre de catégories dans les données.

```
from keras.utils.np_utils import to_categorical

Y_train = to_categorical(y_train, num_classes=6)
Y_train.shape
```

```
(28273, 6)
```

```
[ ]  Y_test = to_categorical(y_test, num_classes=6)
     Y_test.shape
```

```
(7067, 6)
```

Modélisme :

- ### VGG16 :

Création du modèle :

Cette partie du code crée une instance du modèle VGG16 sans les poids pré-entraînés, en excluant la couche de classification finale, et en spécifiant la forme des images en entrée. Ensuite, il affiche un résumé du modèle.

```
[ ]  from keras.applications.vgg16 import VGG16

     base_model = VGG16(
             weights=None,
             include_top=False,
             input_shape=IMG_SIZE + (3,)
         )

     base_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 48, 48, 3)]       0

 block1_conv1 (Conv2D)       (None, 48, 48, 64)        1792

 block1_conv2 (Conv2D)       (None, 48, 48, 64)        36928

 block1_pool (MaxPooling2D)  (None, 24, 24, 64)        0

 block2_conv1 (Conv2D)       (None, 24, 24, 128)       73856

 block2_conv2 (Conv2D)       (None, 24, 24, 128)       147584

 block2_pool (MaxPooling2D)  (None, 12, 12, 128)       0

 block3_conv1 (Conv2D)       (None, 12, 12, 256)       295168

 block3_conv2 (Conv2D)       (None, 12, 12, 256)       590080

 block3_conv3 (Conv2D)       (None, 12, 12, 256)       590080

 block3_pool (MaxPooling2D)  (None, 6, 6, 256)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Définition d'un modèle de réseau de neurones qui utilise le modèle VGG16 pré-entraîné comme base, ajoute des couches entièrement connectées pour l'apprentissage de relations non linéaires, utilise une couche de dropout pour régulariser le modèle et termine par une couche de sortie avec une fonction d'activation softmax pour la classification multiclasse.

```
[ ]  NUM_CLASSES = 6

     model = Sequential()
     model.add(base_model)
     model.add(Flatten())
     model.add(Dense(1000, activation="relu"))
     model.add(Dropout(0.4))
     model.add(Dense(NUM_CLASSES, activation="softmax"))
```

Compilation et entrainement de model :

La fonction suivante, compile et entraîne le modèle, en utilisant les données d'entraînement spécifiées et les hyperparamètres donnés. Elle retourne l'historique d'entraînement du modèle pour une analyse ultérieure.

```
def deep_model(model, X_train, Y_train, epochs, batch_size):

    model.compile(
    loss='binary_crossentropy',
    optimizer=RMSprop(learning_rate=1e-4),
    metrics=['accuracy'])

    history = model.fit(X_train
                        , Y_train
                        , epochs=epochs
                        , batch_size=batch_size
                        , verbose=1)
    return history
```

L'entraînement du modèle avec 40 epochs et une batch_size=64 (exemples d'entraînement qui sont propagés à travers le modèle en une seule passe (ou itération)) :

```
epochs = 40
batch_size = 64

history = deep_model(model, X_train, Y_train, epochs, batch_size)
```

```
Epoch 1/40
442/442 [==============================] - 39s 62ms/step - loss: 0.4619 - accuracy: 0.2477
Epoch 2/40
442/442 [==============================] - 27s 60ms/step - loss: 0.4469 - accuracy: 0.2506
Epoch 3/40
442/442 [==============================] - 27s 61ms/step - loss: 0.4077 - accuracy: 0.3382
Epoch 4/40
442/442 [==============================] - 28s 64ms/step - loss: 0.3673 - accuracy: 0.4209
Epoch 5/40
442/442 [==============================] - 29s 65ms/step - loss: 0.3413 - accuracy: 0.4689
Epoch 6/40
442/442 [==============================] - 28s 64ms/step - loss: 0.3218 - accuracy: 0.5122
Epoch 7/40
442/442 [==============================] - 28s 64ms/step - loss: 0.3029 - accuracy: 0.5524
Epoch 8/40
442/442 [==============================] - 28s 64ms/step - loss: 0.2850 - accuracy: 0.5875
Epoch 9/40
442/442 [==============================] - 28s 64ms/step - loss: 0.2657 - accuracy: 0.6255
Epoch 10/40
442/442 [==============================] - 28s 64ms/step - loss: 0.2458 - accuracy: 0.6617
Epoch 11/40
442/442 [==============================] - 28s 64ms/step - loss: 0.2260 - accuracy: 0.6956
Epoch 12/40
442/442 [==============================] - 28s 64ms/step - loss: 0.2029 - accuracy: 0.7315
Epoch 13/40
442/442 [==============================] - 28s 64ms/step - loss: 0.1805 - accuracy: 0.7672
Epoch 14/40
442/442 [==============================] - 28s 64ms/step - loss: 0.1582 - accuracy: 0.8012
Epoch 15/40
442/442 [==============================] - 28s 64ms/step - loss: 0.1384 - accuracy: 0.8309
```

➔ L'exactitude s'augmente après chaque époque.

Matrice de confusion :

De plus, la sensibilité, la spécificité, le score et la précision sont calculés pour chaque classe, en utilisant la matrice de confusion calculée suivante pour les six classes de détection des émotions. Chaque classe est utilisée contre toutes les classes afin de trouver les facteurs de performance qui lui sont liés.

La fonction suivante, prend en entrée une matrice de confusion, les noms des classes, ainsi que des paramètres optionnels pour la normalisation et le titre de la figure. Elle génère ensuite une représentation visuelle de la matrice de confusion avec des couleurs et des étiquettes appropriées.

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Matrice de confusion',
                          cmap=plt.cm.Blues):

    plt.figure(figsize = (6,6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    cm = np.round(cm,2)
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('Véritable étiquette')
    plt.xlabel('étiquette prédite ')
    plt.show()
```

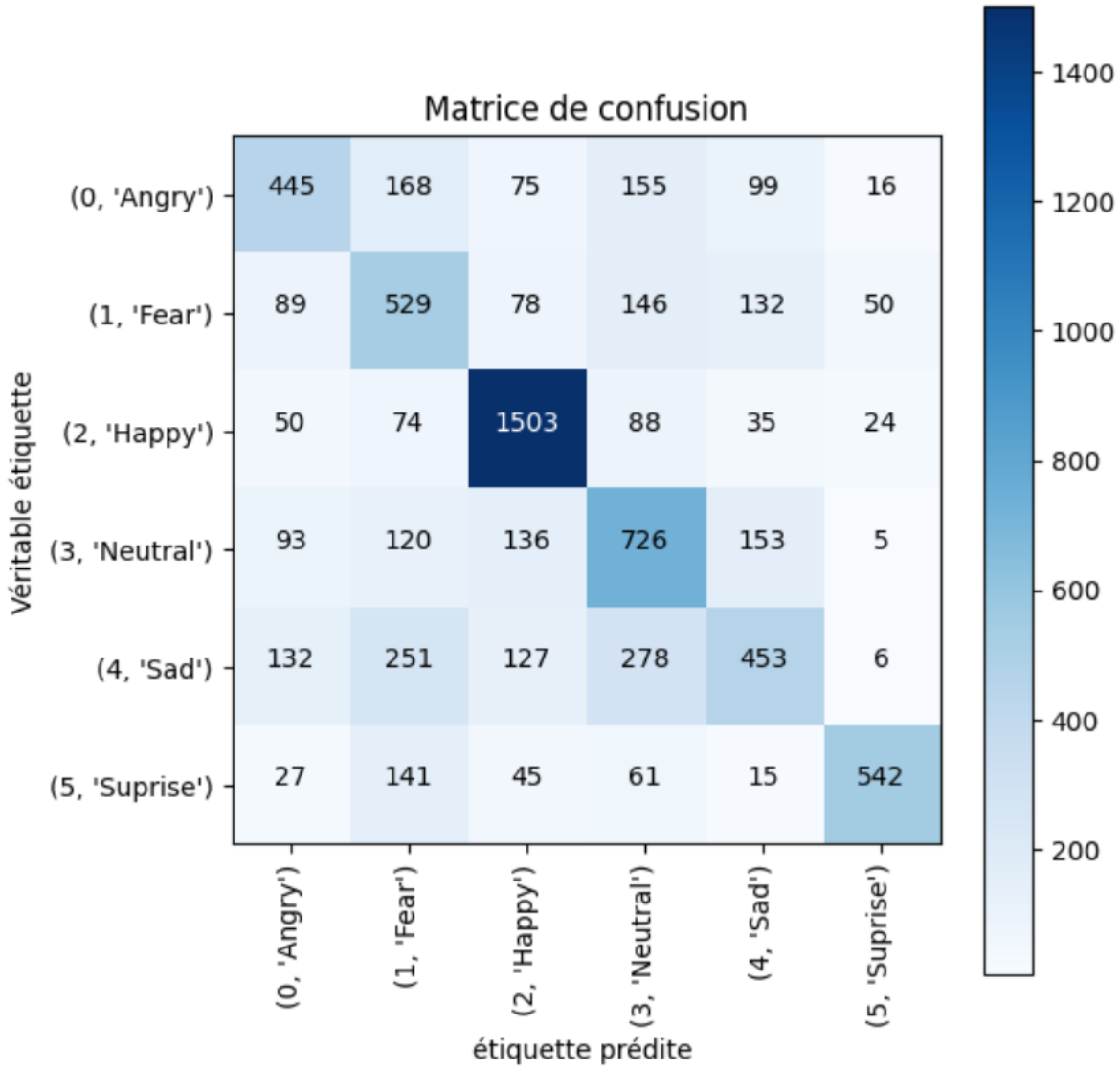
```
# Valider sur jeu de test

predictions = model.predict(X_test)
y_pred = [np.argmax(probas) for probas in predictions]


accuracy = accuracy_score(y_test, y_pred)
print('Précision du test = %.2f' % accuracy)

confusion_mtx = confusion_matrix(y_test, y_pred)
cm = plot_confusion_matrix(confusion_mtx, classes = list(test_labels.items()), normalize=False)
```

```
Précision du test = 0.59
```



D'après la visualisation des prédictions du modèle par rapport aux vraies étiquettes des données, Nous constatons que le modèle a correctement prédit 4 198 échantillons sur 7 067échantillons de test sur, de sorte que le modèle a une précision de plus de 50 %, et ça ce qu'on a trouvé lorsqu'on calcule l'accuracy (59%).

- ## ResNet50:

Création du modèle :

Cette partie du code crée un modèle de base (base model) en utilisant l'architecture ResNet50 pré-entraînée sur l'ensemble de données Training.

```
[ ] base_model = tf.keras.applications.ResNet50(input_shape=(48,48,3),include_top=False,weights="Training/Training/")
```

Le modèle de base ainsi créé peut ensuite être utilisé comme point de départ pour construire un modèle complet en ajoutant des couches supplémentaires personnalisées selon les besoins de la tâche spécifique.

Geler les couches :

Le code suivant, gèle (freeze) les couches du modèle de base, à l'exception des 4 dernières couches.

```
[ ]  # Freezing Layers

     for layer in base_model.layers[:-4]:
         layer.trainable=False
```

En gelant les couches, on empêche leurs poids d'être modifiés, ce qui peut être bénéfique dans plusieurs cas :

- Lorsque le jeu de données sur lequel on entraîne le modèle est petit, en gelant les couches du modèle de base, on évite le risque de surajustement (overfitting) et on permet au modèle d'apprendre des représentations plus générales sur les couches ajoutées.

-Lorsque le modèle de base est déjà très performant sur la tâche cible, en gelant les couches, on peut préserver les connaissances préalables du modèle de base et se concentrer uniquement sur l'entraînement des couches ajoutées pour améliorer les performances spécifiques à la nouvelle tâche.

En gelant les couches du modèle de base, on peut réduire le temps d'entraînement nécessaire, car seules les couches ajoutées nécessiteront des calculs d'optimisation. Cependant, les 4 dernières couches sont marquées comme entraînables, ce qui signifie que leurs poids seront mis à jour lors de l'entraînement du modèle sur les nouvelles données. Cela permet d'adapter ces dernières couches aux spécificités de la nouvelle tâche.

Construction du modèle :

Ce modèle est construit en utilisant le modèle de base ResNet50 pré-entraîné comme extracteur de caractéristiques, suivi de plusieurs couches denses avec des activations ReLU, des normalisations par lot et des couches de dropout pour régulariser le modèle. La couche

finale utilise l'activation softmax pour produire les probabilités des classes pour chaque exemple d'entrée.

```
# Building Model

model=Sequential()
model.add(base_model)
model.add(Dropout(0.5))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(6,activation='softmax'))
```

```
# Model Summary

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2, 2, 2048)        23587712

 dropout_1 (Dropout)         (None, 2, 2, 2048)        0

 flatten_1 (Flatten)         (None, 8192)              0

 batch_normalization (BatchN  (None, 8192)             32768
 ormalization)

 dense_2 (Dense)             (None, 32)                262176

 batch_normalization_1 (Batc  (None, 32)               128
 hNormalization)

 activation (Activation)     (None, 32)                0

 dropout_2 (Dropout)         (None, 32)                0

 dense_3 (Dense)             (None, 32)                1056

 batch_normalization_2 (Batc  (None, 32)               128
 hNormalization)

 activation_1 (Activation)   (None, 32)                0

=================================================================
Total params: 23,885,350
Trainable params: 1,335,782
Non-trainable params: 22,549,568
_________________________________________________________________
```

Compilation et entrainement de model :

Le code suivant, compile le modèle en spécifiant l'optimiseur, la fonction de perte et les métriques à utiliser lors de l'entraînement.

```
model.compile(optimizer='Adam', loss='categorical_crossentropy',metrics=METRICS)
```

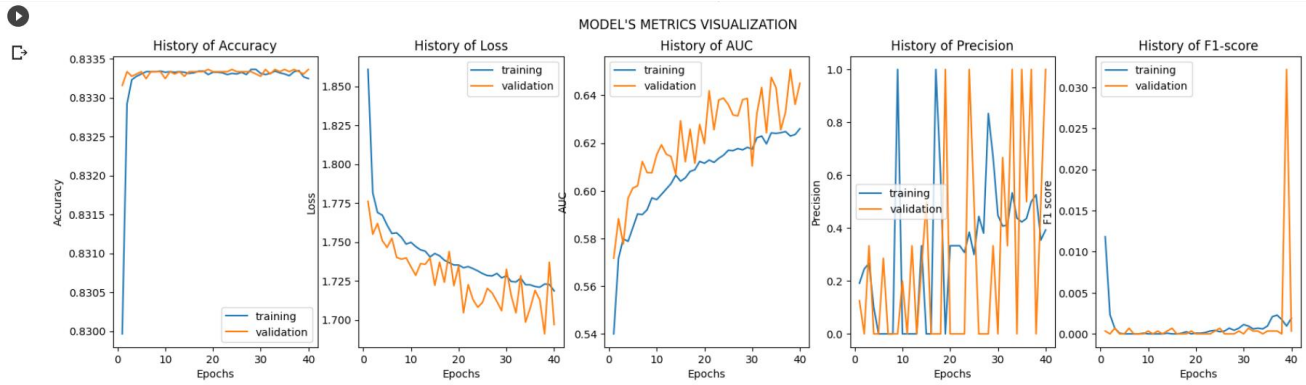
Ce code entraîne le modèle en utilisant les données d'entraînement et de validation fournies, sur un certain nombre d'époques spécifié (dans ce cas, 40 époques). Les détails de l'entraînement, tels que la perte et les métriques, sont affichés avec verbose=1

```
history=model.fit(train_dataset,validation_data=valid_dataset,epochs = 40,verbose = 1,callbacks=[lrd,mcp,es])

Epoch 1/40
354/354 [==============================] - 52s 122ms/step - loss: 1.8609 - accuracy: 0.8300 - precision: 0.1916 - recall: 0.0063 - auc: 0.5401 - f1_
Epoch 2/40
354/354 [==============================] - 39s 110ms/step - loss: 1.7817 - accuracy: 0.8329 - precision: 0.2455 - recall: 0.0012 - auc: 0.5717 - f1_
Epoch 3/40
354/354 [==============================] - 39s 110ms/step - loss: 1.7690 - accuracy: 0.8332 - precision: 0.2667 - recall: 3.5368e-04 - auc: 0.5798 -
Epoch 4/40
354/354 [==============================] - 39s 109ms/step - loss: 1.7673 - accuracy: 0.8333 - precision: 0.1000 - recall: 4.4211e-05 - auc: 0.5788 -
Epoch 5/40
354/354 [==============================] - 39s 110ms/step - loss: 1.7609 - accuracy: 0.8333 - precision: 0.0000e+00 - recall: 0.0000e+00 - auc: 0.584
Epoch 6/40
354/354 [==============================] - 39s 111ms/step - loss: 1.7555 - accuracy: 0.8333 - precision: 0.0000e+00 - recall: 0.0000e+00 - auc: 0.596
Epoch 7/40
354/354 [==============================] - 39s 111ms/step - loss: 1.7559 - accuracy: 0.8333 - precision: 0.0000e+00 - recall: 0.0000e+00 - auc: 0.596
Epoch 8/40
354/354 [==============================] - 39s 111ms/step - loss: 1.7531 - accuracy: 0.8333 - precision: 0.0000e+00 - recall: 0.0000e+00 - auc: 0.592
Epoch 9/40
354/354 [==============================] - 41s 116ms/step - loss: 1.7487 - accuracy: 0.8333 - precision: 1.0000 - recall: 4.4211e-05 - auc: 0.5970 -
Epoch 10/40
354/354 [==============================] - 39s 111ms/step - loss: 1.7498 - accuracy: 0.8333 - precision: 0.0000e+00 - recall: 0.0000e+00 - auc: 0.596
Epoch 11/40
354/354 [==============================] - 39s 111ms/step - loss: 1.7471 - accuracy: 0.8333 - precision: 0.0000e+00 - recall: 0.0000e+00 - auc: 0.598
Epoch 12/40
354/354 [==============================] - 41s 116ms/step - loss: 1.7450 - accuracy: 0.8333 - precision: 0.0000e+00 - recall: 0.0000e+00 - auc: 0.600
Epoch 13/40
354/354 [==============================] - 40s 114ms/step - loss: 1.7441 - accuracy: 0.8333 - precision: 0.0000e+00 - recall: 0.0000e+00 - auc: 0.601
Epoch 14/40
354/354 [==============================] - 42s 118ms/step - loss: 1.7405 - accuracy: 0.8333 - precision: 0.3333 - recall: 4.4211e-05 - auc: 0.6065 -
Epoch 15/40
354/354 [==============================] - 39s 110ms/step - loss: 1.7426 - accuracy: 0.8333 - precision: 0.0000e+00 - recall: 0.0000e+00 - auc: 0.604
```

Les courbes :

Visualisation des résultats de l'entraînement du modèle. On trace plusieurs courbes pour les métriques d'entraînement et de validation, y compris l'exactitude (accuracy), la perte (loss), l'AUC, la précision (precision) et le score F1.



Les sous-graphiques comprennent :
Accuracy : l'exactitude du modèle au fil des époques.
Loss : la perte du modèle au fil des époques.
AUC : l'aire sous la courbe ROC (Receiver Operating Characteristic) du modèle au fil des époques. C'est une mesure de la performance globale du modèle.
Precision : la précision du modèle au fil des époques.
F1-score : le score F1 du modèle au fil des époques. Le score F1 est une mesure de la précision et du rappel (recall) combinés.

En affichant ces courbes, on peut observer la progression de la performance du modèle pendant l'entraînement et évaluer son comportement en termes de surapprentissage ou de sous-apprentissage.

Conclusion Générale

En conclusion, la reconnaissance des expressions faciales à partir d'images est un domaine de recherche prometteur et important dans le domaine de la vision par ordinateur et de l'intelligence artificielle. Comprendre les émotions et les intentions humaines à partir des expressions faciales peut avoir de nombreuses applications pratiques, allant de l'analyse du comportement humain à l'interaction homme-machine.

Au fil des années, des avancées significatives ont été réalisées dans ce domaine, grâce à l'utilisation de techniques d'apprentissage automatique, en particulier les réseaux de neurones convolutifs et l'apprentissage profond. Ces techniques permettent d'extraire des caractéristiques discriminantes des images faciales et de les utiliser pour classer et reconnaître différentes expressions émotionnelles.

Cependant, la reconnaissance des expressions faciales à partir d'images reste un défi complexe en raison de la variabilité des expressions faciales, des conditions d'éclairage, des occlusions, et des différences individuelles. De plus, l'interprétation des expressions faciales peut être subjective et dépendante du contexte culturel.

Pour progresser dans ce domaine, il est nécessaire de continuer à collecter des ensembles de données diversifiés et bien annotés, permettant de capturer la variabilité des expressions faciales dans différentes conditions. De plus, le développement de techniques d'apprentissage automatique plus avancées, telles que l'apprentissage en profondeur avec des architectures de réseaux neuronaux plus complexes, peut contribuer à améliorer les performances de la reconnaissance des expressions faciales.

En fin de compte, la reconnaissance des expressions faciales à partir d'images a le potentiel d'avoir un impact significatif dans de nombreux domaines, tels que la psychologie, les soins de santé, la sécurité et les interfaces homme-machine. Cependant, il reste encore beaucoup de travail à faire pour développer des modèles plus précis, robustes et interprétables, capables de comprendre et d'interpréter pleinement la richesse des expressions faciales humaines.