Mémoire :

Reconnaissance d’émotions par l'expression faciale

Majeure : Intelligence Artificielle

Equipe :

- M.Lucas Barrot
- M.Simon Garras

Référent :

- M. Larbi Boubchir

Avril - Septembre 2020

Table des matières

1. Introduction

1.1. Objectif et équipe

L'objectif principal de ce projet sera de mettre en œuvre une application d'intelligence artificielle qui permettra, à partir de la photo du visage d'une personne, de reconnaître de manière automatique son état émotionnel.
Notre groupe est composé de Lucas Barrot et Simon Garras. Nous sommes tous deux étudiants en 4ème année d'école d'ingénieur, à l'ESME Sudria, dans la majeure Intelligence Artificielle.
Ce projet se place donc dans le contexte de la majeure *intelligence artificielle* dans les branches machine learning et traitement d'image, dans le but d'approfondir nos compétences dans ces deux domaines.

1.2. Informations transmises par le visage

Le visage est porteur de plusieurs informations importantes dont les deux principales sont l'identité et les expressions faciales. Nous ne nous intéresserons ici qu'au expressions faciales car c'est le sujet de notre projet.
Les expressions faciales sont un des moyens les plus utilisés pour transmettre les états émotionnels qui jouent un rôle fondamental dans les interactions entre les personnes.
Les expressions sont déterminées par l'activation des muscles faciaux (comme illustré avec la figure 1.2.1). De plus, certains processus émotionnels peuvent faire changer localement la couleur de la peau, en la colorant localement par un afflux sanguin plus important qu'à l'accoutumée.

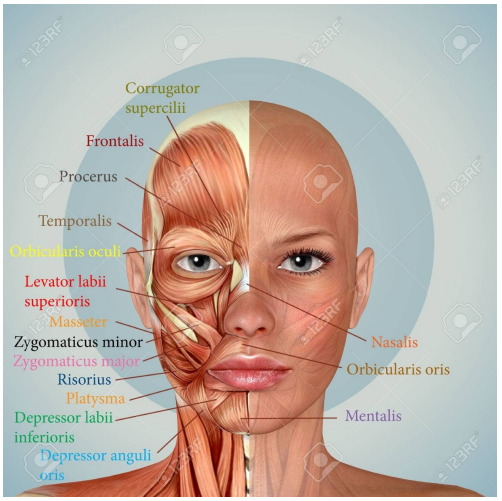


Figure 1.2.1 : Muscles faciaux

C’est à ces expressions, leur nature et les informations qu’elles transmettent que nous allons nous intéresser dans ce projet.

1.3. Problématique

La problématique de ce projet est le développement d’un outil de reconnaissance d’émotions faciales grâce à un algorithme d’intelligence artificielle. Il faut donc choisir un certain nombre d’émotions communes à tout humain et les donner à un algorithme afin qu’il puisse les reconnaître avec le maximum de précision possible.

1.4. Domaines d’applications

Apprendre à reconnaître l’état émotionnel d’une personne peut rendre possible la simulation de ses états sur une machine et améliorer l’interaction entre l’homme et la machine.

On peut également imaginer d’autres domaines d’application comme dans la sécurité ou encore le commerce. En effet, reconnaître un visage et son expression peut également amener à interpréter ses actions présentes (et potentiellement futures) à des fins de prévention (risque terroriste, fraude, fatigue ou manque d’attention en voiture), à des fins commerciales (Avoir directement le retour du client sur un produit en analysant son expression faciale) ou simplement publicitaires (On peut imaginer une publicité adaptée à l’émotion perçue d’un potentiel client).

1.5. Plan du mémoire

Nous allons détailler dans le prochain chapitre dans un premier temps comment fonctionnent les émotions et comment on peut les reconnaître et les interpréter, puis faire un état de l’art des méthodes de reconnaissance émotionnelle déjà existantes en s’intéressant plus particulièrement aux méthodes utilisant le deep learning pour démontrer leur efficacité. Puis pour finir nous aborderons des améliorations et des perspectives de développements possibles du projet.

2. Reconnaissance des émotions

Avant de pouvoir parler de la reconnaissance des émotions, il faut revenir à ce qu'est une émotion. Nous allons ensuite parler des grandes théories servant à catégoriser les émotions

2.1. L'émotion

L'émotion a été défini sous quatre grands aspects : physiologique, subjectif, cognitif et expressif.

L'aspect physiologique des émotions se manifeste par les changements corporels accompagnant le changement de l'état subjectif comme le changement de la température corporelle, de la fréquence cardiaque et respiratoire. La détection de ces aspects nécessite des mesures avec des capteurs liés aux sujets, ce qui s'avère être intrusif.

L'aspect subjectif des émotions qui est lié au ressenti par les individus.

L'aspect cognitif qui est lié à la compréhension de l'individu sur une scène ou un événement.

L'aspect expressif qui est constitué des expressions faciales, des expressions corporelles et des intonations vocales.

Notre projet s'inscrit dans le cadre d'une reconnaissance des émotions à travers leurs aspects mesurables à savoir ici non pas l'aspect physiologique (donc sans l'utilisation d'outils intrusifs) ou l'aspect subjectif ou cognitif (qui relèvent plus de la psychologie), mais bien de l'aspect expressif

Il faut à présent dresser une liste des émotions "affichable" par un visage et (plus important) les catégoriser.

2.2. Séquencement des émotions

De nombreuses études se sont portées sur les émotions et leurs séquencement.

On peut noter la théorie de l'universalité des émotions qui suppose que l'émotion est universelle, dotée de fonctions adaptatives et de fonctions communicatives. Elle a été exposée par Darwin dans son livre "L'expression des émotions chez l'homme et les animaux".

Selon sa théorie, les émotions sont imprimées dans le système nerveux humain, les rendant universelles. Il classe les changements des expressions faciales en sept groupes pour lesquels les déformations du visage y sont décrites de façon détaillée.

Cette théorie a engendré de nombreuses études qui ont tenté de la prouver en travaillant sur un nombre restreint d'émotions renommées émotions de base (émotions discrètes, émotions primaires ou émotions fondamentales).
D'après Ekman, les émotions de base ont des caractéristiques uniques et leurs réactions sont préprogrammées. De plus, elles sont souvent représentées comme des émotions innées et indépendantes des cultures. Ekman définit sept autres critères pour caractériser les émotions fondamentales [3], tels que la présence de la même émotion chez un autre primate ou la présence de caractéristiques physiologiques qui la distinguent des autres émotions

La théorie de l'universalité a facilité la représentation des émotions, notamment la représentation discrète (catégorielle).

La représentation discrète catégorise certaines émotions dans des groupes prédéfinis ayant des caractéristiques distinctes les unes des autres. Ce sont les émotions de bases qui symbolisent ces catégories.

Les chercheurs qui adoptent le concept des émotions de base présentent un ensemble restreint d'émotions, mais leurs recherches divergent lorsqu'il s'agit de les identifier. Les travaux ci-dessous présentent les émotions de base selon chaque auteur :

- Ekman : colère, peur, tristesse, joie, dégoût, surprise.
- Tomkins : colère, dégoût, joie, peur, surprise, mépris, honte, intérêt, anxiété.
- Izard : colère, surprise, dégoût, joie, peur, tristesse, mépris, intérêt, culpabilité, honte.

Un autre courant de pensées représente les émotions de manière continue (dimensionnelle) basé sur deux ou plusieurs dimensions.

Contrairement à la représentation catégorielle, les émotions sont définies en se basant seulement sur les dimensions. Russell définit les états affectifs sur un modèle bidimensionnel, représenté par un cercle basé sur un axe de plaisir (plaisir/peine) et un axe quantifiant la force du ressenti (illustré sur la Figure 2.2.1).

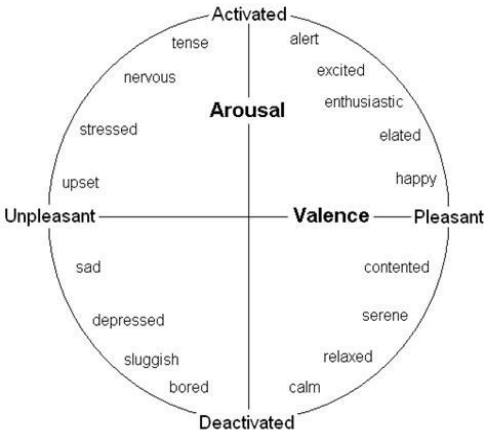


Figure 2.2.1 : Modèle circulaire de Russell

Toutes les émotions peuvent être définies sur ce modèle. Si on prend l'exemple de l'enthousiasme, il s'agit d'un état associant un taux de plaisir élevé à un taux d'activation élevé. D'après Russell, c'est un état qui se trouve à un angle de 45° en considérant que le plaisir est l'angle 0° et l'état d'éveil est l'angle 90°.

D'autre part, Cowie et al proposent un outil pour l'annotation des émotions sur un modèle bidimensionnel très proche de celui de Russell, qui se base sur l'activation et l'évaluation (positif-négatif).

Le modèle de Plutchik une autre manière de représenter les émotions. Huit émotions de base (colère, dégoût, peur, joie, tristesse, surprise, acceptation et anticipation) sont représentées sur un cercle (sans pour autant les définir par des dimensions). Des émotions secondaire sont définies par des combinaisons des émotions de base adjacentes. Une dimension d'intensité détermine également l'intensité des émotions de base (Figure 2.2.2).

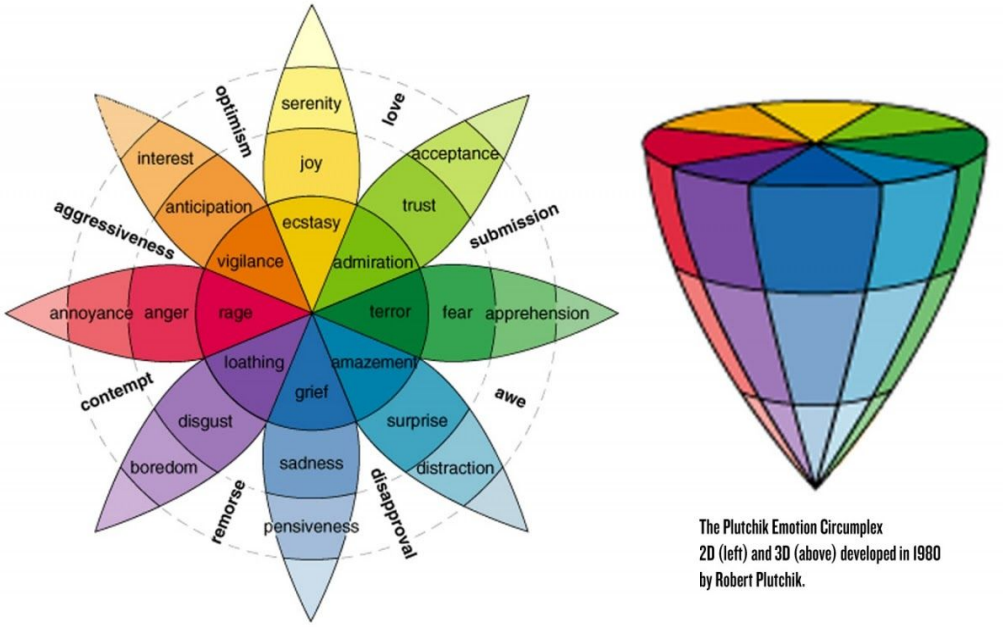


Figure 2.2.2 : Modèle de Plutchik

3. État de l'art: Reconnaissance d’émotions

Nous avons choisi de diviser ce chapitre en deux, car nous nous sommes rendu compte que les méthodes les plus efficaces entraient dans le cadre du deep learning.
Cependant, il est intéressant de parler de méthodes (d’une en particulier) qui n'entrent pas dans ce cadre-là et qui ont déjà été utilisées pour faire de la reconnaissance d’émotion par l’expression faciale.

3.1. Sans deep learning : La méthode SVM (Machines à Vecteurs de Support)

Les SVMs sont très utilisés pour résoudre des problèmes de discrimination, c'est-à-dire décider à quelle classe appartient un échantillon, ou de régression, c'est-à-dire prédire la valeur numérique d'une variable. Ils ont rapidement été adoptés pour leur capacité à travailler avec des données de grandes dimensions et le faible nombre d'hyper-paramètres.

Classification entre deux catégories

Un SVM classe les données d’entrainements dans un SVC (Vector Support Classifier ou Classeur à vecteur de support) en déterminant un hyperplan optimal qui sépare ces données par rapport à leur catégorie.

Pour trouver un hyperplan optimal, il faut définir deux autres notions qui sont la marge et le vecteur support.

- Le vecteur support est le vecteur orthogonal à l’hyperplan qui a pour norme la distance entre l’hyperplan et les données classées la plus proche de celui-ci.
- La marge est l’épaisseur de l’espace proche de l’hyperplan qui ne contient pas de données. Autrement dit, la marge correspond à la norme du vecteur support

On parle de d'hyperplan optimal quand la marge entre les classes est la plus grande (car les erreurs de prédictions sont minimisées).
A noter que les SVMs prennent en compte la souplesse de la marge. Autrement dit, des données peuvent volontairement être ignorées pour avoir un meilleur hyperplan.

La figure 3.1.1 présente des hyperplans possibles entre deux classes de données. Le graphique de gauche présente des hyperplans non-optimaux tandis que le graphique de droite présente l’hyperplan optimal soit avec la marge la plus grande.

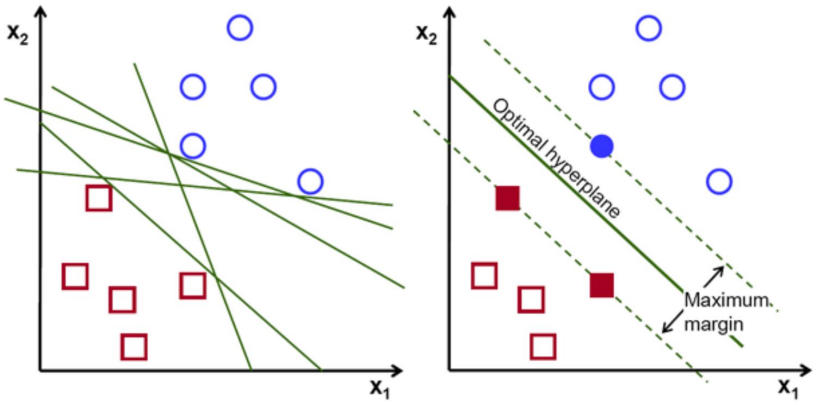


Figure 3.1.1 : Différents hyperplans possibles contre l'hyperplan optimal

Si la distribution des données n'est pas linéaire, la stratégie des SVMs consiste à augmenter la dimension des données jusqu'à ce que les données soient linéairement séparables et trouve un hyperplan optimal pour classer les données dans un SVC. La valeur de la nouvelle dimension de chaque données est donnée grâce à une fonction mathématique appelée "fonction kernel".

Classification entre plusieurs catégories (dans le cas de notre étude, les catégories correspondes aux émotions se dégageant du visage présent sur l'image)

Pour résoudre un problème de classification à deux classes, il s'agit de trouver l'hyperplan le plus éloigné des points les plus proches de chaque classe.
Quand le nombre de classes augmente, la stratégie des SVMs consiste à ramener le problème en plusieurs classifications à deux classes. Il existe deux approches.

La première approche consiste à créer autant de classeur qu'il y a de classes possibles et chaque classeur donnera une probabilité d'appartenance d'une nouvelle donnée à une des classes plutôt qu'aux autres regroupement des autres classes. Une fois la nouvelle donnée passée dans tous les classeurs, elle est classée dans la catégorie prédite avec la probabilité la plus élevée. Cette méthode se nomme One-VS-All ou One-VS-Rest.

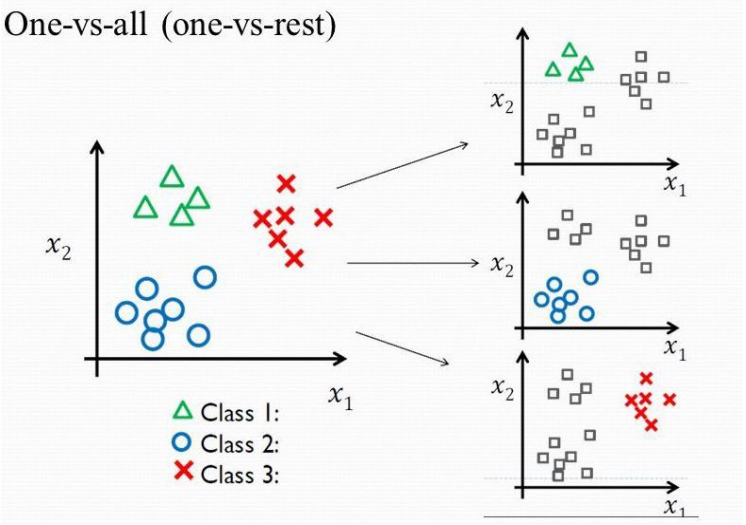


Figure 3.1.2 : Multi-classification avec l'approche One-VS-All

La seconde approche consiste à créer autant de classeur qu'il y a de paire de classes possibles et chaque classeur évaluera si chaque donnée appartient plus à une ou à une autre des classes. Chaque classeur prédit un une catégorie pour la donnée et c'est la catégorie qui a reçu le plus grand nombre de vote qui reçoit la nouvelle donnée. Cette méthode se nomme One-VS-One .

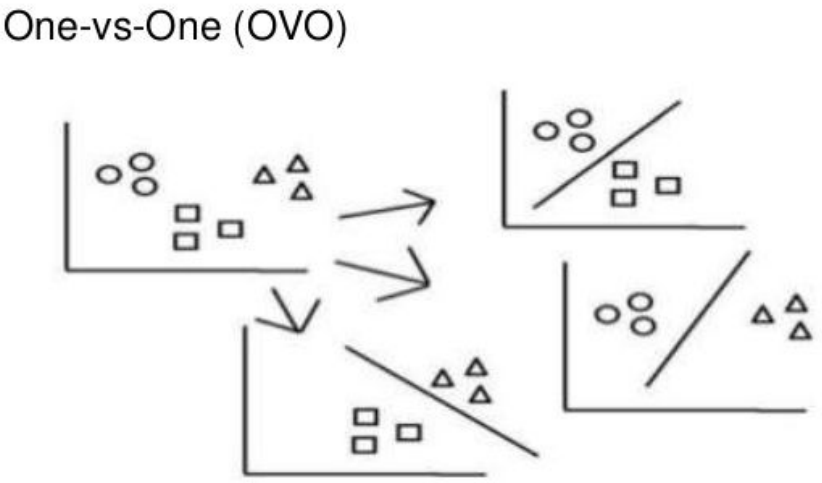


Figure 3.1.3 : Multi-classification avec l'approche One-VS-One

La méthode d'utilisation des SVM est basiquement la même que pour les autres méthodes d'apprentissage supervisé. On crée le modèle, on l'entraîne, (on le test et on l'évalue,) et on l'utilise pour prédire les nouvelles valeurs.

Le principal problème d'utiliser la svm pour classifier les émotions dégagées par une image est que l'accuracy trouvée est relativement faible (Accuracy de 0.37).

C'est pourquoi nous nous sommes tournées vers les méthodes de deep learning.

3.2. Avec Deep learning

3.2.1. L'ajout du deep learning par rapport au supervisé

La méthode support vector machine (SVM) est utilisée principalement pour des problèmes de classification et de régression. La plupart du temps, la SVM sera utilisée sur des jeux de données réduits et des problèmes linéaires la où les méthodes deep learning et plus particulièrement les réseaux de neurones convolutionnels (CNN) vont prouver leur efficacité en utilisant des jeux de données très grands et sur des problèmes plus complexes.

Le plus gros avantage du deep learning est donc sa capacité d'adaptation à des problèmes plus complexes et sa capacité à utiliser des jeux de données bien plus grands, en effet, on peut augmenter la complexité d'un réseau de neurones en ajoutant des couches à l'intérieur du réseau et/ou en augmentant leur taille, alors qu'en SVM on ne peut pas du tout augmenter la complexité du modèle, cela se traduit néanmoins par un temps d'entraînement plus long et plus coûteux en ressource.

Cependant dans notre contexte nous allons montrer qu'une méthode SVM sera donc insuffisante car notre problème est très complexe et a beaucoup de paramètres à prendre en compte, la taille et la forme du visage, les micro-expressions varient en fonction des personnes et de plus on cherche à reconnaître une émotion parmi 6. La tâche est donc relativement complexe, Les prochain chapitre seront donc sur les méthodes utilisées.

3.2.2. Les différents réseaux neuronaux :

Le deep learning ou l'apprentissage profond utilise les réseaux de neurones artificiels pour résoudre des problèmes généralement complexes et non-linéaires

RNN

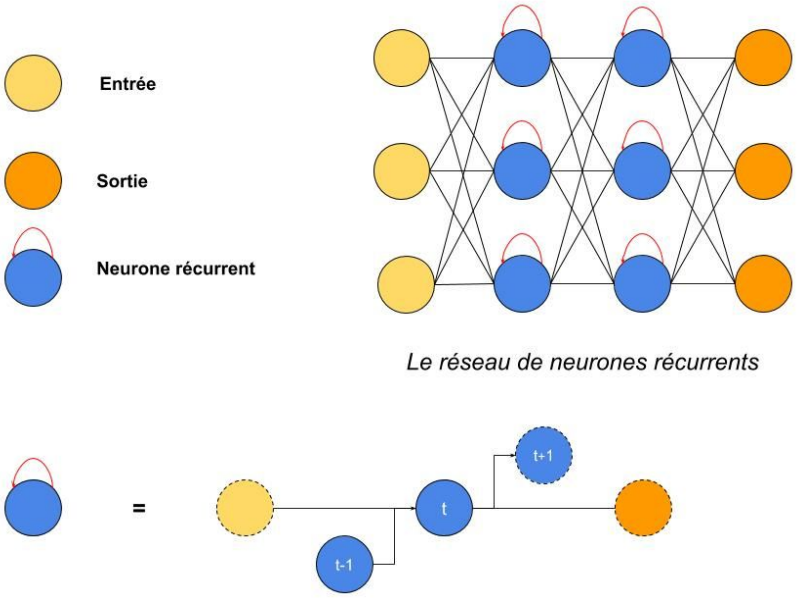
Les réseaux de neurones récurrents (RNN) sont largement utilisés en intelligence artificielle dès lors qu'une notion temporelle intervient dans les données.
Ils sont largement utilisés dans l'analyse de texte (prédiction, traduction, reconnaissance automatique), dans l'analyse vocale ou encore dans la reconnaissance de formes. ils sont très similaires aux réseaux de neurones artificiels (ANN) :

Un ANN fonctionne comme tel (voir Figure 3.2.2.1) :

Des données d'entrées (input) arrivent dans la couche d'entrée du réseau. Les données sont sous la forme d'un vecteur, par exemple (0.7, 0.4, 0.9) et chaque coordonnée du vecteur est envoyée aux neurones d'entrée (jaunes).
Ensuite, les 3 valeurs vont avancer dans le réseau couche par couche (1ère bleue puis 2nde bleue puis orange qui est la sortie). Les couches bleues sont dites cachées et la orange dite couche de sortie.
Pour avancer, entre chaque neurone il y a un trait qui les relie : ce trait est associé à une valeur dite le poids (par exemple 0.3 entre le 1er neurone jaune et le 1er bleu) qui va

pondérer la valeur entrante (i.e. on aura 0.7*0.3 = nouvelle valeur qui arrive dans le 1er neurone bleu).
Toutes les valeurs entrantes (et pondérées) sont additionnées à l'entrée d'un neurone puis on applique une certaine fonction au résultat, ce qui donne une valeur de sortie pour chaque neurone.
Ces valeurs sont ensuite propagées à la couche suivante et ainsi de suite.

Contrairement à un ANN (et c'est là la seule différence), sur chaque neurone bleu, on a une boucle (développée sous le schéma du réseau) :
On a en entrée un vecteur composé de 3 valeurs, une pour chaque neurone jaune, elles se propagent ensuite dans le réseau à la manière d'un ANN, sauf que chaque neurone bleu reçoit en plus des sorties des neurones de la couche précédente, sa propre sortie si il avait en entrée la valeur que le neurone précédent a eu en entrée. de plus sa valeur de sortie est conservée et servira pour le prochain neurone.
On a donc une mémoire de 1 itération pour les neurones bleus.



Le réseau de neurones récurrents

Figure 3.2.2.1 : Schéma représentatif d'un RNN

Cette méthode souffre néanmoins d'un problème majeur qui est son temps d'apprentissage, en effet, comme beaucoup d'autres méthodes, le calcul et la mise à jour des poids se fait grâce à la méthode de rétropropagation du gradient, or en RNN la fonction la plus utilisée pour ce calcul est la fonction tanh qui donne en sortie une valeur entre 0 et 1. On obtient donc très vite plus on recule dans le réseau des valeurs proches de 0, ce qui signifie que les neurones des couches précédentes ont très vite un impact très faible sur la modification des poids, le réseau peut donc facilement "oublier" des données prises en compte plus tôt, on dit qu'il a la mémoire courte.

CNN

Les réseaux de neurones convolutifs ou réseau de neurones à convolution (en anglais CNN) sont des réseaux de neurones artificiels feedforward, c'est à dire que la transmission de données dans le réseau est unidirectionnelle, dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Les neurones de cette région du cerveau sont arrangés de sorte qu'ils correspondent à des régions qui se chevauchent lors du pavage du champ visuel. Leur fonctionnement est inspiré par les processus biologiques, ils consistent en un empilage multicouche de perceptrons, dont le but est de prétraiter de petites quantités d'informations.

Un réseau de neurones CNN est composé principalement de deux types de neurones organisés en couches qui traitent successivement l'information :

- Les neurones de convolution :
  Ces neurones ont pour travail d'effectuer un traitement convolutif sur une portion de l'image limitée au moyen d'une fonction de convolution.
- Les neurones de mise en commun :
  Ces neurones, dits de "pooling" ont pour travail de regrouper les différentes portions de l'image après qu'elles aient été traitées par le noyau de convolution afin qu'elles soient transmises à la couche suivante.

Une architecture CNN est composée d'un empilement de couche de traitements dont les principales sont :

- la couche de convolution qui traite les données d'un champ récepteur.
- la couche de pooling, qui permet de compresser l'information en réduisant la taille de l'image intermédiaire (souvent par sous-échantillonnage)
- la couche de correction (ReLU), souvent appelée par abus « ReLU » en référence à la fonction d'activation (Unité de rectification linéaire).
- la couche « entièrement connectée » (FC), qui est une couche composée de perceptrons.

Ci-dessous un schéma explicatif d'une architecture CNN très commune.

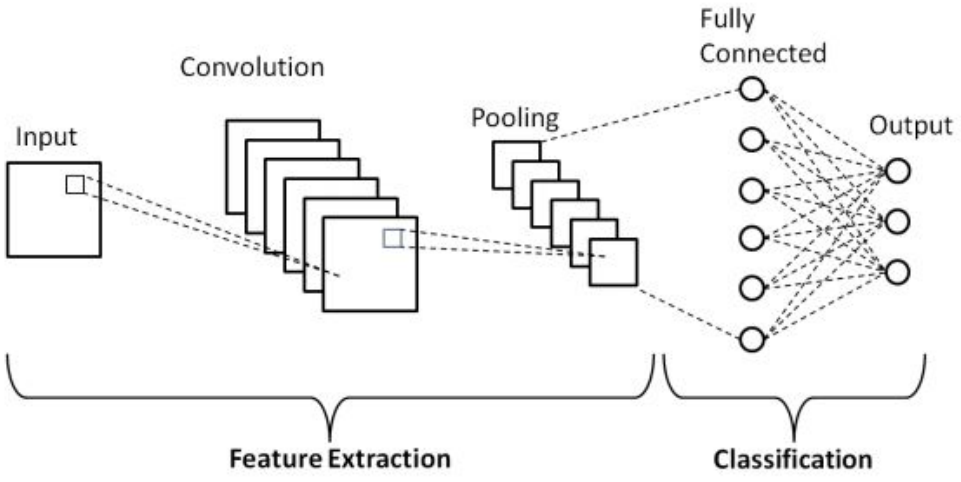


Figure 3.2.2.2 : Schéma représentatif d'un CNN

Le principal avantage des CNN pour la résolution de notre problème est que ceux-ci sont construits pour et peuvent augmenter en complexité “simplement” en rajoutant des couches intermédiaires afin d’avoir un modèle plus profond et plus performant.
Il existe des centaines d'architectures différentes de CNN, nous allons donc en choisir quelques unes et les entraîner sur notre jeu de données afin de voir laquelle est la plus adaptée et nous renvoie les meilleurs résultats.

Auto encoding

De manière général : le principe d’un auto encodeur est d’apprendre à compresser et encoder les données puis apprendre à les reformer le plus fidèlement possible à partir de la forme compressée (comme illustré dans la **figure 3.2.2.2)** .
De cet manière, l’auto encodeur apprend à ignorer le bruit.

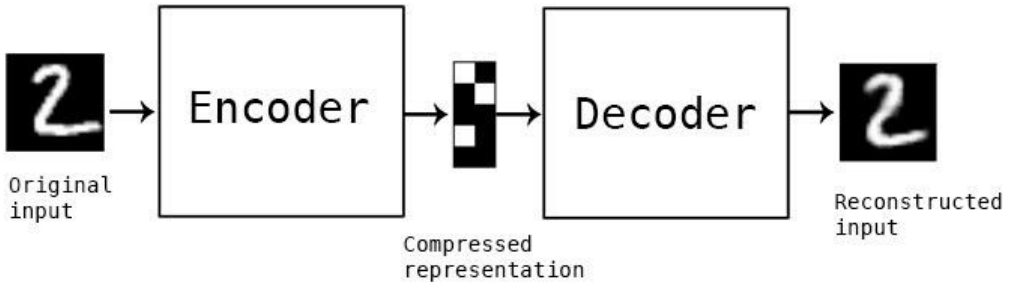


Figure 3.2.2.2 : Schéma illustratif du fonctionnement d’un auto-encodeur

Ce réseau de neurones est très utilisé pour la détection et la réduction de bruit **(figure 3.2.2.3)** ou la détection d’une anomalie.

Le réseau de neurones d’un auto encodeur est constitué de 4 grandes parties

1. L’encodeur : Dans cette partie, le model apprend à réduire les dimensions de la donnée d’entrée et la compresse dans une forme dite “encodée”.
2. Le Goulot : qui est l'entité qui contient la plus petite version de la donnée encodée.
3. Le décodeur: Dans cette partie, le model apprend à reconstruire la donnée originelle en se basant uniquement sur la donnée contenue dans le goulot.
4. La perte d’information: Qui est la méthode qui mesure la qualitée de la reconstruction.

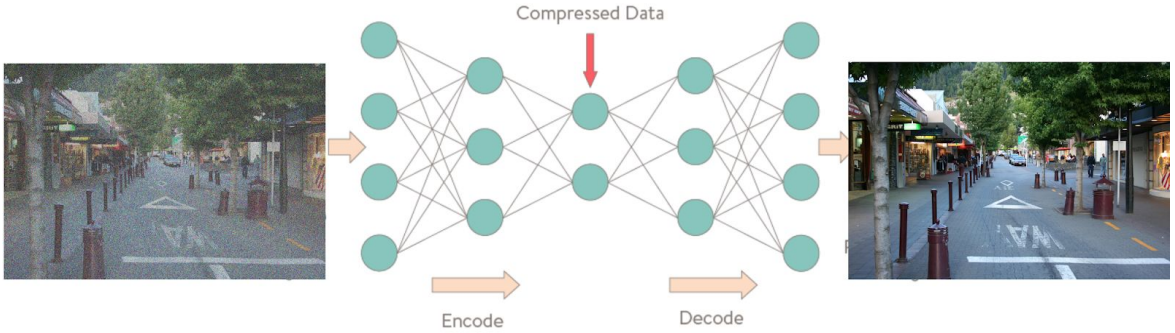


Figure 3.2.2.3 : Schéma exemple du model de l’auto encoding pour la réduction de bruit

4. Approche Proposée :

Suite à notre étude des différents moyen existants pour répondre à notre problématique, Nous allons mettre en place deux méthodes que nous allons ensuite comparer, tout d'abord nous allons utiliser une méthode d'apprentissage profond qui est la méthode CNN, puis nous allons utiliser la méthode d'apprentissage supervisé SVM et nous allons les comparer.

4.1. Les Ressources utilisées

4.1.1. Le dataset

Le dataset que nous avons choisi est le dataset du challenge FER 2013 qui est un dataset qui a déjà largement été utilisé dans la recherche sur la reconnaissance d'émotion sur des visages par des algorithmes. Nous allons utiliser le même dataset pour entraîner les deux modèles afin d'avoir une meilleure comparaison.
Il est composé de 35 887 images réparties de la manière suivante :

- 4953 Images de visage ayant l'expression : Colère
- 547 Images de visage ayant l'expression : Dégoût
- 5121 Images de visage ayant l'expression : Peur
- 8989 Images de visage ayant l'expression : Joie
- 6077 Images de visage ayant l'expression : Tristesse
- 4002 Images de visage ayant l'expression : Surprise
- 6198 Images de visage ayant l'expression : Neutre

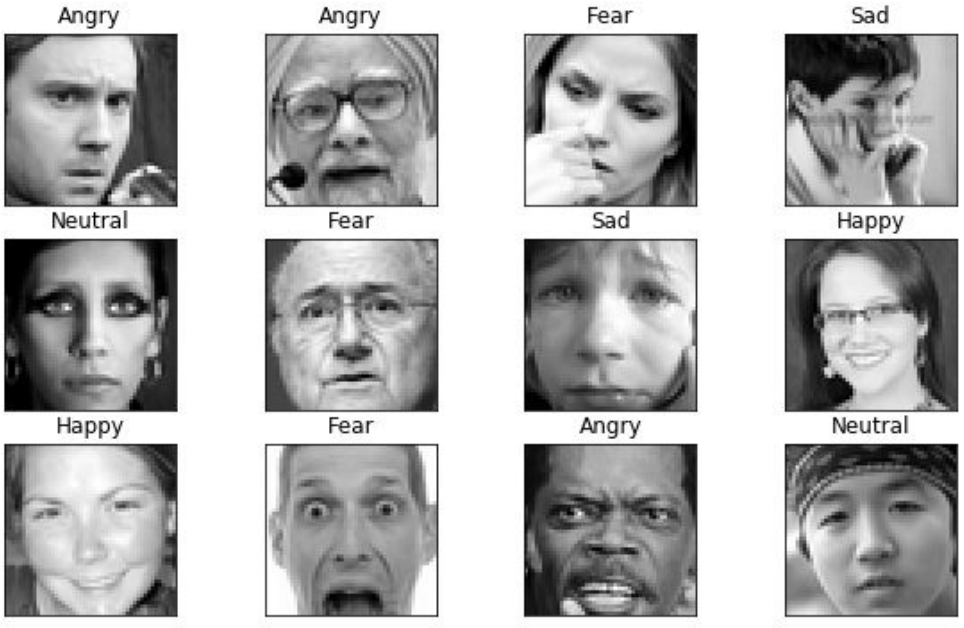


Figure 4.1.1. Exemples d'images du dataset

Ce dataset est assez complet pour nous permettre d'entraîner nos modèles, cependant l'émotion "dégoût" est clairement sous représentée et même avec de l'augmentation de données il est compliqué de combler le manque d'images, de plus l'émotion est difficilement différenciable de la peur ou de la colère avec lesquelles elle est grandement confondue. nous avons donc décidé de l'enlever à cause de son manque de pertinence.

```
data = data.drop(data[data.emotion == 1].index)
data.loc[data['emotion'] == 6, 'emotion'] = 1
```

Ici on supprime toutes les images ayant pour emotion la valeur 1 qui correspond au dégoût, Puis on attribue aux images ayant pour emotion la valeur 6 la valeur 1 pour avoir des valeurs de 0 à 5.

4.1.2. Préparation des données pour le CNN

Pour préparer ces données afin d'entraîner le modèle nous avons commencé par ouvrir le fichier et l'avons associé à un objet "dataframe" de pandas :

```
data = pd.read_csv("fer2013.csv")
```

Puis on initialise les variables qui vont contenir les données d'entraînement :

```
#initialisation des conteneurs :
CATEGORIES = ['Angry', 'Neutral', 'Fear', 'Happy', 'Sad', 'Surprise']
X = []
y = []
```

On crée ensuite la fonction qui nous permet d'extraire les pixels et les labels qui leur correspondent afin de transformer la chaîne de caractère pixels d'abord en une liste d'entiers distincts puis en une array à deux dimensions et enfin qui rentre ces données dans une liste "X" pour les images et "y" pour les labels :

```
29  def load_data():
30      print("Loading data...")
31      for index, row in data.iterrows():
32          try:
33              # Restructuration de la chaine de caractères en une liste d'entiers distincts = pixels
34              pixels=np.asarray(list(row['pixels'].split(' ')), dtype=np.uint8)
35              # Structuration des pixels sous la forme d'une array à deux dimensions
36              img = pixels.reshape((48,48))
37              # Ajout de l'image et du label correspondant aux listes correspondantes
38              X.append(img)
39              y.append(row['emotion'])
40          except Exception as e:
41              print(e)
42      print("data loaded")
```

Puis on l'appelle :

```
42  load_data()
```

On transforme ensuite les listes obtenues en np.array pour qu'elles puissent être utilisées dans la fonction train_test_split :

```
# Restructuration des images et des labels en arrays pour qu'elles puissent passer dans la fonction de split
X = np.array(X, dtype='float32').reshape(-1, 48, 48, 1)
X=X/255.
y = np.asarray(y)
```

On sépare les données en données d'entraînement et de validation en gardant 30% des données pour le test, on utilise l'option shuffle pour mélanger les données avant de les séparer et l'option stratify pour garder les mêmes proportion de labels entre l'entraînement et le validation :

```
# Séparation des données pour l'entrainement
(X_train, X_val, y_train, y_val) = train_test_split(X, y, test_size=0.3, random_state=5, shuffle=True, stratify=y)
```

On utilise l'ImageDataGenerator de keras pour étoffer le dataset en générant des nouvelles images à partir de celles existantes.
Puis on l'appelle :

```
# Augmentation des données pour améliorer les performances du modèle
train_data_aug = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")

validation_gen = ImageDataGenerator()
```

Pour finir on normalise les données d'entraînement en les passant dans les fonctions d'augmentation de données créées plus tôt :

```
validation_gen = ImageDataGenerator()
# Normalisation des données d'entrainement
train_data_aug.fit(X_train)
validation_gen.fit(X_val)
```

4.1.3. Modèle CNN

Pour notre modèle CNN on a choisi d'utiliser comme outil de construction python avec les librairies de tensorflow et keras.

```
# Construction du modèle :

model = Sequential()
```

La construction du modèle commence par sa définition, ici on le définit en séquentiel, c'est a dire qu'il va être sous la forme d'une succession de couches par lesquelles l'information va passer dans un seul sens à savoir de l'entrée à la sortie : model = Sequential()

Couche d'entrée :

```
model.add(Conv2D(32, (3, 3), input_shape=(48,48,1), padding='same'))
```

La première couche qui est la couche d'entrée a pour paramètres :

- La taille de celle ci, qui est le nombre de neurones qu'elle contient. on choisit en général des puissances de deux, ici on a choisi 32.
- La taille de la matrice de convolution, autrement appelé kernel. Pour des images en dessous de 128*128 (ici on a 48*48), il est recommandé de ne pas utiliser de kernel supérieur à 3 par 3 pour ne pas réduire trop vite la dimension de l'information
- Le format d'entrée, ici la taille et le nombre de couleurs de l'image (ici une seule car l'image est en niveaux de gris).

- Ici le paramètre “stride” n’est pas modifié, de par la taille relativement réduite de l’image, on n’as pas besoin d’accélérer la réduction de dimension au risque de perdre de l’information
- puis pour finir le “padding”, qui n’as que deux paramétrages possibles à savoir “same” ou “valid” ici on choisit “same” car on veut préserver les dimensions d’entrées

```
model.add(Activation('relu'))
```

Vient après la définition de la fonction d’activation de la couche, ici on a choisi la fonction “relu” pour Rectified Linear Unit car c’est la plus adaptée pour notre modèle.

```
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
```

Pour finir la première couche on ajoute une opération de “pooling” qui va passer une matrice sur l’image convoluée de la taille définie par “pool_size” afin de garder les valeurs les plus élevées et donc les plus influentes. Ici on garde toujours le padding à “same” pour conserver les dimensions de l’image.

Couches cachées :

```
# Couches cachées
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
```

Se suivent ensuite 4 couches cachées ainsi définies, elles sont différentes de l’entrée quant à leur taille, et elles sont constituée d’une opération supplémentaire :

```
model.add(BatchNormalization())
```

La Batch Normalization permet de normaliser les poids qui ont été modifiés pour éviter de déstabiliser l’entraînement en ayant des poids qui prendraient une trop grande importance, cela permet aussi d’augmenter la rapidité de l’entraînement.

Couche dite “fully connected” ou FC :

```
102     model.add(Flatten())
103     model.add(Dense(512))
104     model.add(Activation('relu'))
105     model.add(Dropout(0.2))
```

La couche FC est la couche qui sert d’intermédiaire entre les opérations successives de convolution et la couche de sortie, elle est composée d’abord d’une fonction “Flatten” qui permet de convertir les données sortant des couches de convolution (qui sont des arrays à trois dimensions) en une liste à une seule dimension afin de les passer dans une couche dite “Dense” qui est une couche ayant autant de perceptrons que la dernière couche de convolution (ici 512 perceptrons) qui ont pour fonction d’activation la fonction relu. On ajoute une couche “Dropout” afin de limiter le surentraînement du modèle.

Couche de sortie :

```
# Couche de sortie
model.add(Dense(6, activation='softmax'))
# Compilation du modèle
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

La couche de sortie est composée d'une seule couche ayant un nombre de perceptrons égal au nombre d'émotions possible, ici 6.
On compile ensuite le modèle avec comme optimizer "adam" qui est un optimizer très utilisé pour ce genre de modèles, on obtient un modèle ayant 2 622 086 paramètres entraînables, ce qui est largement suffisant pour répondre à notre demande.

Entraînement :

On lance pour finir l'entraînement de notre modèle en utilisant le fit_generator avec une taille de batch de 32 c'est à dire que les images vont passer 32 par 32 dans le modèle, on définit le nombre maximal d'epoch d'entraînement (une epoch définit un passage complet du dataset dans le modèle) que l'on met ici à 100, et enfin on spécifie les callbacks, ici on utilise seulement l'earlystopping que l'on a défini plus haut avec une patience de 7 epochs, ce qui nous permet de ne pas sur-entraîner le modèle en s'arrêtant quand il ne progresse plus.

```
earlystop = EarlyStopping(patience=7)

training = model.fit_generator(train_data_aug.flow(X_train, y_train, batch_size=32),
                    validation_data=validation_gen.flow(X_val, y_val, batch_size=32),
                    steps_per_epoch=len(y_train) // 32,
                    epochs=100,
                    callbacks=[earlystop])

fer_json = model.to_json()
with open("fer.json", "w") as json_file:
    json_file.write(fer_json)
model.save_weights("fer.h5")
```

Puis on sauvegarde le modèle et la valeur des poids une fois l'entraînement fini dans deux fichiers pour les réutiliser plus tard.

4.1.4. Préparation des données pour le SVM

Etant donné que nous utilisons le même dataset pour les deux méthodes, la préparation des données pour la méthode SVM ressemble à celle utilisée pour le CNN.

On commence par ouvrir le fichier et l'avons associé à un objet "dataframe" de pandas :

```
datasetCSVFile = "fer2013.csv"
fer_data = pd.read_csv(datasetCSVFile)
```

Puis on initialise les variables qui vont contenir les données d'entraînement :

```
CATEGORIES = ['Angry', 'Neutral', 'Fear', 'Happy', 'Sad', 'Surprise']
X = []
y = []
```

On crée ensuite la fonction qui nous permet d'extraire les pixels et les labels qui leur correspondent afin de transformer la chaîne de caractère pixels en une liste d'entiers

distincts. On rentre ensuite ces données dans une liste “X” pour les “images” (la liste des pixels de chaque image) et “y” pour les labels. Puis on l’appelle

```
def load_fer_data():
    for index, row in fer_data.iterrows():
        try:
            pixels=np.asarray(list(row['pixels'].split(' ')), dtype=np.uint8)
            X.append(pixels)
            # img = pixels.reshape((48,48)) #In the case of data visualisation
            # X.append(img)
            y.append(row['emotion'])
        except Exception as e:
            pass

load_fer_data()
```

On laisse les données sous forme d’une liste à une dimension pour respecter les dimensions d’entrée acceptées par la fonction d'entraînement du modèle SVM.

On transforme ensuite les listes obtenues en np.array pour qu’elles puissent être utilisées dans la fonction train_test_split :

```
X = np.array(X, dtype='float32')
X=X/255.
y = np.asarray(y)
```

On sépare les données en données d'entraînement et de validation en gardant 30% des données pour le test, on utilise l’option shuffle pour mélanger les données avant de les séparer et l’option stratify pour garder les mêmes proportion de labels entre l’entraînement et le validation :

```
(X_train, x_test, y_train, y_test) = train_test_split(X, y,
                                                      test_size=0.3,
                                                      random_state=42,
                                                      shuffle=True,
                                                      stratify=y)
```

4.1.5. Modèle SVM

On crée le modèle comprenant les hyperparamètres pour son entraînement.

```
model = svm.SVC(kernel='rbf', gamma='auto', decision_function_shape='ovr', C=0.8)
```

Le paramètre “Kernel” correspond à la forme que prendra la fonction qui va calculer la relation entre chaque données (et simulera l’augmentation dimension). On a choisi ‘rbf’ pour ‘radian based function’ soit fonction radiale et son paramètre gamma qui représente la valeure du coefficient de la fonction kernel. Laisser gamma à ‘auto’ signifie donner à gamma la valeure inverse au nombre de features qui composent les données.
Le paramètre ‘décision_function_shape’ correspond à l’approche utilisé pour résoudre le problème de classification à plusieurs classes. On choisit ‘ovr’ pour l’approche One-VS-Rest.
Enfin, le paramètre C correspond à la souplesse de la marge (plus petit est C, plus souple est la marge soit un plus grand nombre de valeurs seront ignorés dans la recherche d’hyperplan entre deux classes.

On entraîne le modèle avec les paramètres définis lors de sa définition sur les données d'entraînement.

```
model.fit(X_train, y_train)
```

4.2. Résultats

CNN :

Nous avons testé plusieurs niveaux de complexité différents atteints en augmentant progressivement la taille des couches cachées.

Nous avons aussi testé plusieurs tailles de batch et sommes rapidement arrivés à la conclusion que 32 était un bon compromis, en effet les temps d'entraînement étaient considérablement rallongés avec des tailles différentes sans apporter d'amélioration des résultats.

Nous avons testé l'optimizer "rmsprop" qui est similaire à "adam" mais nous n'avons pas remarqué de changements significatif donc nous avons gardé "adam".

Enfin nous avons testé l'entraînement avec et sans l'émotion "disgust" et nous pouvons voir que les performances sont meilleures et plus régulières sans cette émotion.

Nous avons gardé tout au long de ces tests une patience égale à 7 epochs car au vu de l'évolution de l'entraînement des modèles cette valeur semble bonne, en effet l'augmenter n'aidera pas les modèles car au delà de 30 epochs environ le modèle ne progresse presque plus mais la diminuer en revanche pourrait augmenter les chances qu'un modèle s'arrête trop tôt à cause d'une série d'époch un peu longue sans améliorations.

Batch_size	Patience	Optimizer	Trainable params	Time per epoch (sec)	Nb of epochs trained	val_accuracy	val_loss	Total training time (min)	number of output
16	7	adam	147015	48	68	0,619	1,0206	54,4	7
64	7	adam	147015	56	47	0,6056	1,0522	43,86666667	7
32	7	adam	147015	45	35	0,5854	1,098	26,25	7
32	7	adam	455303	51	33	0,6046	1,0315	28,05	7
32	7	adam	1243527	66	34	0,6149	1,0367	37,4	7
32	7	adam	2622599	100	30	0,5951	1,0849	50	7
32	7	adam	147015	43	50	0,6168	1,0043	35,83333333	6
32	7	adam	455303	48	38	0,6132	1,0088	30,4	6
32	7	adam	1243527	56	41	0,6237	0,9784	38,26666667	6
32	7	adam	2622599	98	47	0,6319	0,9701	72,85	6

Figure 4.2.1. Tableur résumant l'entraînement des différents modèles CNN.

On atteint donc avec notre modèle le plus performant 63% de précision sur les prédictions, ce qui est un score plutôt bon au vu des performances actuelles en terme de reconnaissance des émotions et en prenant en compte nos moyens.

On a pu cependant constater que l'augmentation de la complexité des modèles en terme de nombre de paramètres entraînables n'est pas proportionnelle à l'augmentation de ses résultats, en effet on gagne environ 1,5 % de précision en plus pour un modèle ayant 15 fois plus de paramètres entraînable et ayant besoin de plus de deux fois plus de temps d'entraînement.

5. Utilisation des résultats pour le développement d'un web service :

Architecture de l'application reconnaissance d'émotions :

L'application est composée de trois services distincts :

- Une application mobile avec ionic 5 : Elle récupère l'image via l'appareil photo ou la bibliothèque d'image du téléphone à la suite de quoi elle envoie l'image sur notre serveur.
- Le serveur développé avec Flask (une librairie python pour créer des API) et hébergé sur Heroku : il reçoit l'image de l'application, la pré-traite (notamment en la centrant sur le visage, en rognant et en changeant sa résolution), et l'envoie au modèle afin qu'elle soit traitée par l'algorithme. Celui-ci renverra, à la suite de ce traitement, un message indiquant émotion prédite par l'algorithme
- L'Api de Computer Vision de Microsoft Azure dans laquelle est envoyée l'image et qui retourne la position du visage dans l'image.

5.1. L'application ionic

L'application ionic est ce qui est le plus proche de l'utilisateur. En effet, c'est l'utilisateur qui va prendre la photo ou en choisir une sur son album et l'envoyer au serveur pour l'analyser.

Ionic est un framework qui permet de développer une application mobile en se basant sur le modèle du développement web.
l'application se matérialise par un stack(tas ou pile) de pages à travers lesquelles l'utilisateur navigue grâce au routing d'angular.
le framework combine les langages suivants :

- Angular : angular est la base de la construction de l'application, il permet d'effectuer tout le routing de l'application (navigation interne et externe).
- HTML/CSS : chaque page contient un fichier html et un fichier css pour définir ce qui sera affiché sur cette page
- Typescript : chaque page contient un fichier typescript qui contient toute la logique d'exécution de la page

En plus de ces langages, Ionic est basé sur Cordova qui permet d'ajouter un grand nombre de plug-ins permettant notamment de gérer toutes les applications natives des mobiles sur lesquels l'application est installée comme par exemple la caméra, la librairie de fichiers, la lampe etc…
Cela fait donc de Ionic un outil complexe, mais très puissant pour développer rapidement des applications avec des fonctionnalités natives pour mobile.
A noter que le choix de ionic comme Framework de développement permet d'avoir une solution multiplateforme (Android, iOS et Desktop) et adaptative en fonction de la plateforme sur laquelle elle est installée.

L’application est composée d’une page d’accueil elle même composée d’un texte de présentation, d’un cadre pour accueillir l’image qui sera envoyée, et de trois bouton servant respectivement à sélectionner une Image dans la galerie du téléphone, prendre une Image avec l’appareil photo, et envoyer le résultat au serveur.

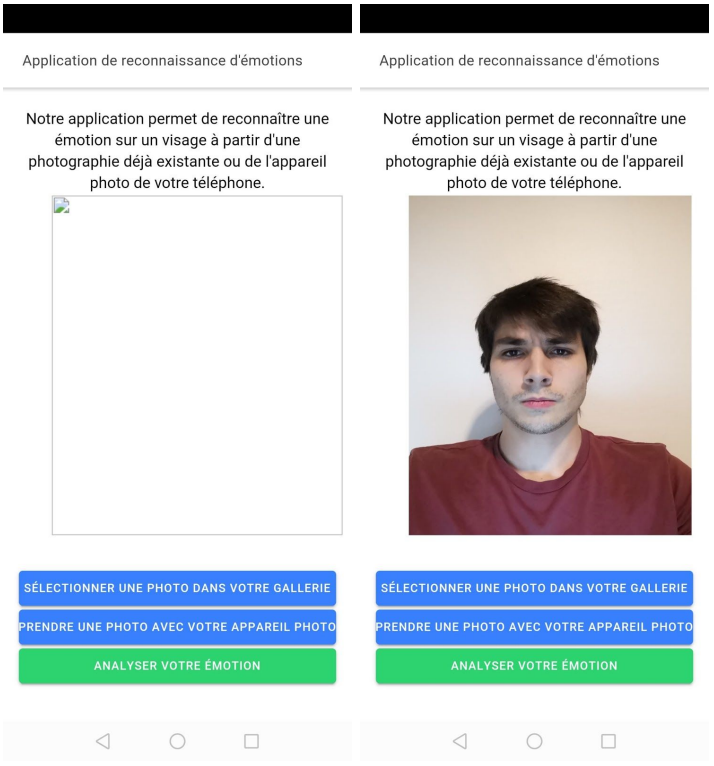


Figure 5.1.1. Captures d’écran de l’application Mobile

5.2. Le serveur

Le serveur permet de recevoir l’image pour la traiter puis l’analyser.

Il reçoit l’image envoyée par l’application, la préretraite à l’aide de deux fonctions.
La première détecte les visages présents sur l’image et renvoie sous forme de json les coordonnées du premier visage rencontré.

Plus précisément, cette fonction envoie une requête à l’api REST de Computer Vision de Microsoft Azure (nous en parlons dans la partie 4.2.3). La requête demande à l’outil d’Azure d’analyser l’image envoyé (qu’on utilise au format <byte array>) en précisant dans les paramètres que l’on souhaite récupérer les informations sur les visages présents sur l’image. La réponse contient effectivement des informations sur les visages de l’image dont leurs positions ainsi qu’un identifiant qui leurs est associés. A la suite de quoi la fonction ne garde que les informations de la position et la taille du premier visage rencontré et renvoie ces informations à la suite du programme.

Après avoir détecté un visage dans l’image, celle-ci va être recadrée et redimensionnée par une seconde fonction de sorte que le visage soit centré et que l’image respecte les

dimensions demandés par le modèle de reconnaissance d'émotion (48px*48px). Bien entendu, la fonction renvoie l'image retouché (au format PIL.Image.Image)

Une fois ces deux fonctions effectuées, le modèle est chargé puis on lui passe l'image retouchée pour qu'il l'analyse.
Le résultat de l'analyse est envoyé à l'application mobile sous la forme d'un objet json contenant une variable "prédiction" qui à pour valeur la prédiction du modèle.

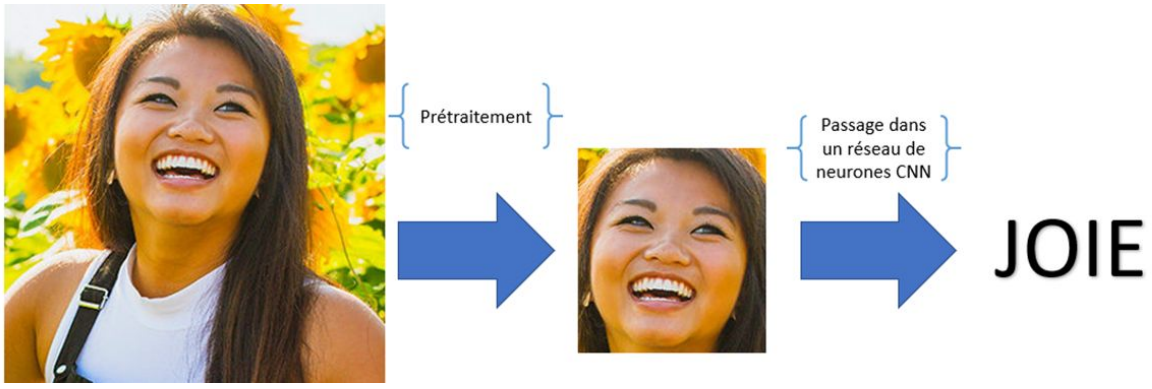


Figure 5.2.1. Schéma du traitement de l'image après réception par le serveur

5.3 L'Api REST Computer Vision de Microsoft Azure

Computer Vision est un service qui donne accès à des algorithmes avancés permettant de traiter des images et de retourner des informations en fonction des caractéristique visuelles qui intéressent. Par exemple, Computer Vision peut déterminer si l'image est prise en intérieur ou en extérieur, identifier des marques ou des objets spécifiques ou encore, ce qui nous a intéressé, trouver des visages humains.
Il est possible de créer des applications Computer Vision par le biais d'un kit SDK de bibliothèque de client ou en appelant directement l'API REST (ce que nous avons utilisé).

Le service Computer Vision est disponible à l'utilisation dans la plate-forme applicative en cloud de Microsoft : Microsoft Azure.
En se créant un compte gratuit de 30 jours, on peut accéder à de nombreux services gratuitement. On place les services désirés dans des groupes de ressources, on récupère la clé d'authentification et le point de terminaison (différents pour chaque service) afin que par la suite, on puisse communiquer avec l'api du service auquel on souhaite accéder.

5.4 Fonctionnalités additionnelles et particularité de notre application

Nous avons créé un GitHub afin de mieux nous organiser pour la mise en place et le déploiement de l'application. Le lien : https://github.com/LucasBarrot/FER2020.

Il est également important de noter que le serveur mis en place n'est pas géré en local. Il est hébergé par une plateforme d'application en cloud de la société Heroku, qui permet d'héberger son serveur web gratuitement (sous condition d'un respect d'une limite d'espace occupable par le serveur).

6. Conclusion

Ce projet nous a permis de découvrir les méthodes d'apprentissage profond et ses difficultés et de nous rendre compte de la difficulté qu'un algorithme aussi complexe qu'un réseau de neurones peut avoir à reconnaître un visage et une émotion.
Nous avons cependant pu remarquer que les méthodes plus anciennes telles que le SVM sont bien moins performantes à ressources égales.

Ce projet nous a permis d'avoir un avant goût des difficultés liées à l'utilisation de plusieurs langages de développement différents pour une même application.

Les objectifs principaux à savoir le développement d'une méthode d'apprentissage profond pour effectuer de la reconnaissance d'émotions faciale ont été atteints mais l'objectif de développement d'un web service reliant application mobile et serveur n'as pas été atteint, en effet il y a un problème dans la liaison entre le serveur et l'application.

Dans l'optique d'un projet de 5ème année, nous pourrions créer une application mobile qui intégrerait tous ces composants en son sein, c'est à dire que le modèle et toutes les opérations de traitement d'images seraient intégrées dans la structure de l'application, il faudrait donc changer de méthode pour le développement de l'application mobile afin de pouvoir intégrer ces fonctionnalités.

7. Références Bibliographique

émotions :

- https://tel.archives-ouvertes.fr/tel-01622639/document
- https://www.irit.fr/publis/TCI/Dalle/these_mercier.pdf
- https://edutice.archives-ouvertes.fr/edutice-00000702/document
- P. Ekman. Are there basic emotions? Psychological Review, 99 :550–553, 1992
- J. A. Russell. A circumplex model of affect. Journal of personality and social psychology, 39 :1161–1178, 1980.

méthodes :

SVM :

- https://dumas.ccsd.cnrs.fr/dumas-00745988/document
- http://biblio.univ-annaba.dz/wp-content/uploads/2016/09/These-Benmohamed-Abderrahim.pdf
- https://github.com/amineHorseman/facial-expression-recognition-svm
- https://medium.com/@ODSC/build-a-multi-class-support-vector-machine-in-r-abcdd4b7dab6
- https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

CNN vs SVM :

- https://www.quora.com/What-is-the-difference-between-CNN-and-a-support-vector-machine#:~:text=Convolution%20Neural%20Network%20is%20non,model%20complexity%20isn't%20possible.
- https://iopscience.iop.org/article/10.1088/1755-1315/357/1/012035/pdf
- https://www.quora.com/Why-does-the-convolutional-neural-network-have-higher-accuracy-precision-and-recall-rather-than-other-methods-like-SVM-KNN-and-Random-Forest
- https://www.quora.com/Why-is-CNN-better-than-SVM
- https://www.quora.com/What-is-the-difference-between-CNN-and-a-support-vector-machine#:~:text=Convolution%20Neural%20Network%20is%20non,model%20complexity%20isn't%20possible.

Différentes méthodes :

- https://datakeen.co/3-deep-learning-architectures-explained-in-human-language/

RNN :

- https://blog.octo.com/les-reseaux-de-neurones-recurrents-des-rnn-simples-aux-lstm/
- https://penseeartificielle.fr/comprendre-lstm-gru-fonctionnement-schema/#:~:text=Un%20r%C3%A9seau%20de%20neurones%20r%C3%A9currents,tr%C3%A8s%20r%C3%A9pandu%20en%20deep%20learning.&text=Toutes%20les%20valeurs%20entrantes%20(et,de%20sortie%20pour%20chaque%20neurone

CNN :

- https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif#:~:text=En%20apprentissage%20automatique%2C%20un%20r%C3%A9seau,est%20inspir%C3%A9%20par%20le%20cortex
- https://www.researchgate.net/figure/Schematic-diagram-of-a-basic-convolutional-neural-network-CNN-architecture-26_fig1_336805909

Auto encoding :

- https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726
- https://fr.wikipedia.org/wiki/Auto-encodeur

Application mobile :

- https://blog.engineering.publicissapient.fr/2017/07/24/on-device-intelligence-integrez-du-deep-learning-sur-vos-smartphones/