ESIEA

4A – Mars 2023

PROJET Machine Learning Challenge : Reconnaissance d'émotions

RAPPORT

Enseignant
Thibault GEOFFROY

Semestre 8

Étudiants
Derhen PESON, Tom GONIN, Annaly ESCHYLLE, Marwa MOHAMED, Jean-Georges PETRI-GUASCO

2022-2023

Table des matières

Présentation du problème

La reconnaissance d'émotions est un domaine de recherche très prometteur de l'intelligence artificielle. En utilisant par exemple des techniques de Machine Learning, il est possible de construire des modèles qui peuvent apprendre à identifier les émotions d'une personne à partir de données d'entrée, telles que des images, des signaux audios ou des textes. Ces modèles peuvent être utilisés dans de nombreux domaines, tels que l'éducation, la psychologie, la médecine, l'industrie du divertissement et des réseaux sociaux.

Notre projet est donc de proposer une solution en Python pour résoudre le problème de reconnaissance d'émotions. Pour cela nous avons travaillé sur deux types de données d'entrée qui nous ont été mis à disposition :

- Des données textes où nous avons utilisé du Machine Learning
- Des images avec lesquelles nous avons travaillé en Deep Learning

Ce rapport vous présentera les deux approches que nous avons eues pour résoudre ce problème, la démarche associée et les différents résultats obtenus.

Voici les liens vers notre colab (code), gitlab(stockages des fichiers d'entrainement et test)

Lien du gitlab : marwa-mohamed-dev/test_MLChallenge (github.com)

Lien du colab : https://colab.research.google.com/drive/1LcGFmL-RoydSTzcqI59xRDZSb8uggpSq?usp=sharing

Recherche de modèle ML pour exploiter les données du fichier training_data.csv

Description des features et analyse des données

Avant de rentrer dans notre démarche de prédiction, nous avons pris le temps de découvrir le jeu de données qui a été mis à notre disposition

Ainsi le dataset est composé de 140 features et 1287 exemples. Les features comprennent :

- un identifiant : id,
- le numéro correspondant à l'émotion de l'individu sur l'image : target,
- L'émotion de l'individu sur l'image : targetName,
- Numéro du sujet observé : subject,
- 137 caractéristiques correspondant aux points (x,y) importants du visage

Ensuite nous avons pu réaliser deux vérifications : la présence ou non de valeurs manquantes et la répartition de nos classes dans le Dataset.
Ces vérifications sont importantes afin de ne pas rencontrer de problèmes lors de nos prédictions (comme avoir un sur ou sous apprentissage).

Ainsi, nous n'avons pas retrouvé de valeurs manquantes dans le jeu de données et nous avons pu remarquer que les émotions étaient bien réparties dans notre jeu de données avec une représentation de 13 à 15%.

Entrainement

Nous avions d'abord souhaité tester sur nos données, un modèle simple que nous connaissons, le KNN. Le résultat n'étant pas très significatif, nous avons décidé de rechercher un autre modèle de ML.

Nous avons trouvé la librairie Lazypredict qui compare différents modèles et effectue une prédiction et un calcul du score et du temps d'exécution.

Salon la sélection des données de test et d'entrainement par la fonction train_test_split, le résultat peut différer. Toutefois, même si RidgeClassifierCV n'est pas toujours en tête, il reste très souvent parmi les 5 meilleurs modèles avec une différence de score faible et un meilleur temps d'exécution. Nous avons donc choisi d'utiliser le modèle RidgeClassifierCV.

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LogisticRegression	0.77	0.78	None	0.77	0.56
LinearSVC	0.77	0.78	None	0.77	3.38
CalibratedClassifierCV	0.77	0.78	None	0.77	7.94
RidgeClassifierCV	0.74	0.75	None	0.74	0.07
RidgeClassifier	0.74	0.75	None	0.73	0.03
LinearDiscriminantAnalysis	0.71	0.72	None	0.71	0.29

Étant donné que selon la séparation entre les données de test et d'entrainement, le score change, nous avons calculé un score moyen et avons obtenu un score d'environ **71,5%.**

Traitement des données

Afin d'améliorer le score obtenu, il faut rendre les données plus exploitables. Nous avons pensé à effectuer un calcul de distance à partir des coordonnées des points.

Nous avons alors recalculé le score moyen obtenu avec notre modèle (RidgeClassifierCV). Cependant, cela a très faiblement amélioré le score.

Nous avons alors relancé lazypredict afin de potentiellement trouver un modèle plus performant, LinearDiscriminantAnalysis, mais il n'y avait pas d'amélioration importante du score. **Le meilleur score obtenu est de 80,6 %.**

Résultat obtenu à l'aide de LinearDiscriminantAnalysis :

```
Moyenne des Accuracy : 73.73%
Moyenne des temps d'exécution : 0.026395s
Accuracy maximal obtenu : 80.57%
```

Temps

Sans traitement de données, le temps d'exécution du modèle RidgeClassifierCV est en moyenne de 0,14 seconde. Après traitement des données, il est passé à 0.035 secondes.

Résultats obtenus avant traitement des données avec RidgeClassifierCV :

```
Moyenne des Accuracy : 71.555%
Moyenne des temps d'exécution : 0.14117s
Accuracy maximal obtenu : 76.425%
```

Résultats obtenus après traitement des données avec RidgeClassifierCV :

```
Moyenne des Accuracy : 72.214%
Moyenne des temps d'exécution : 0.034311s
Accuracy maximal obtenu : 76.943%
```

Après traitement des données et en utilisant LinearDiscriminantAnalysis, le temps moyen d'exécution du modèle est de 0.026 secondes.

Dans les 3 cas, nous obtenons un temps d'exécution relativement faible, ce qui est une bonne chose en matière d'impact écologique.

Recherche d'hyperparamètres

Le GridSearch est une technique d'optimisation des hyperparamètres qui consiste à tester différentes combinaisons d'hyperparamètres pour trouver la combinaison qui permet d'obtenir les meilleures performances pour un modèle de Machine Learning. Pour rappel les hyperparamètres sont des paramètres qui ne sont pas appris pendant l'entraînement d'un modèle, mais qui sont choisis avant l'entraînement et qui influencent la façon dont le modèle est entraîné comme par exemple le taux d'apprentissage ou encore le nombre d'itérations. En choisissant les bons hyperparamètres, vous pouvez améliorer la précision et la robustesse du modèle. Cependant, il est souvent difficile de déterminer les meilleurs hyperparamètres à utiliser, car il existe de nombreuses combinaisons possibles.

Paradoxalement, en utilisant les hyperparamètres déterminés par le GridSearch, nous n'avons pas trouvé de bien meilleurs résultats. Ceci est surement dû au fait que nous n'avons pas eu assez de temps afin de définir la grille d'hyperparamètres qui allaient être réentraîné dans notre GridSearch.

Utilisation d'un réseau neurone convolutif (CNN) afin de constituer un modèle d'entrainement pour la reconnaissance d'émotion sur des images

Nous avons décidé en parallèle de l'utilisation d'algorithmes de machine learning d'utiliser un CNN pour analyser directement le dataset d'image sans passer par le fichier CSV.

Un réseau neuronal convolutif est un type d'architecture de réseau de neurones profonds utilisé pour l'apprentissage automatique supervisé, en particulier pour l'analyse d'images. Les CNN ont été inspirés par la manière dont le cerveau traite les informations visuelles et sont conçus pour extraire des caractéristiques des images en utilisant des filtres de convolution.

Dans un réseau neuronal convolutif, les images sont entrées dans le réseau sous forme de tenseurs. Le réseau comprend des couches de convolution qui appliquent des filtres de convolution sur les images pour extraire des caractéristiques telles que des contours, des textures, des motifs, etc. Ensuite, des couches de mise en commun (pooling) sont utilisées pour réduire la taille des images en conservant les caractéristiques les plus importantes.

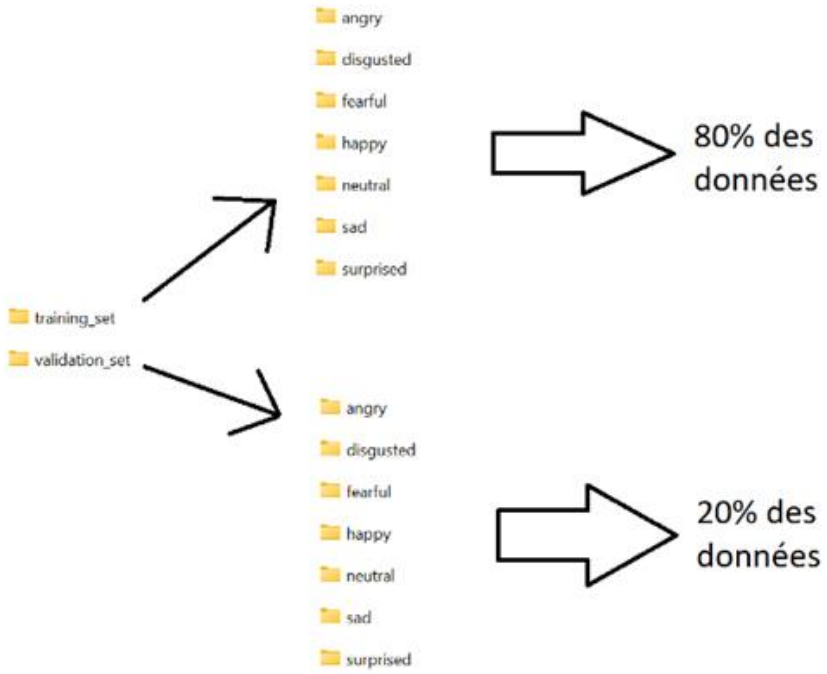
Après plusieurs couches de convolution et de mise en commun, les caractéristiques extraites sont envoyées à une ou plusieurs couches entièrement connectées (fully connected) pour la classification. Ces dernières couches sont utilisées pour prendre une décision finale sur la classe à laquelle appartient l'image.

Les CNN sont largement utilisés pour la reconnaissance d'image car ils sont capables de traiter des images de grande taille et de haute résolution de manière efficace, tout en étant capables d'apprendre

automatiquement les caractéristiques importantes des images sans la nécessité d'une ingénierie de caractéristiques manuelles. Ils ont été utilisés avec succès dans de nombreuses applications, notamment la reconnaissance faciale, la détection d'objets, la reconnaissance de caractères manuscrits, la reconnaissance de texte, etc. C'est pour cette raison que nous avons décidé d'utiliser ce type de réseau de neurones.

Prétraitement et chargement des données :

Avant de charger les données nous avons procédé à la séparation des images en deux datasets séparés : training_set et validation_set. Pour chaque répertoire les images étaient ensuite classées dans 7 dossiers représentant les labels.



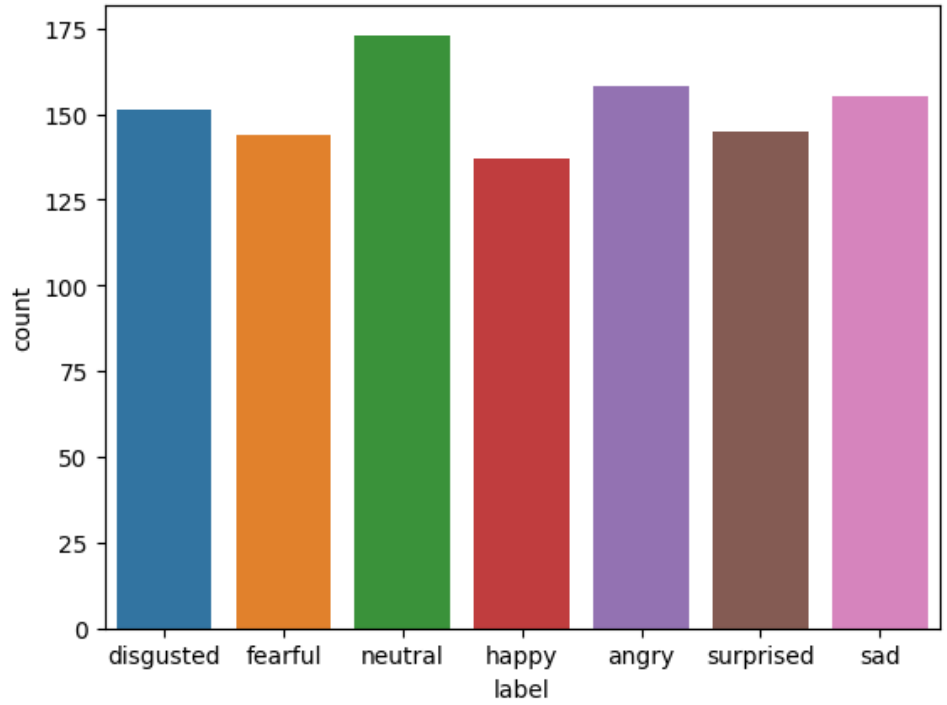
Une fois la création des deux fichiers nous avons chargé chacun des datasets :

```python
# Create a data generator to load images from directories
train_datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical')

val_datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
validation_generator = val_datagen.flow_from_directory(
        test_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical')
```

```
Found 1028 images belonging to 7 classes.
Found 258 images belonging to 7 classes.
```

Graphique présentant le nombre d'images du training_set en fonction des labels :



Nous avons donc 1028 images classées en 7 classes pour le training_set et 258 images classées en 7 classes pour le validation_set.

Le paramètre **rescale=1./255** indique que les valeurs de pixel de l'image doivent être mises à l'échelle pour qu'elles soient comprises entre 0 et 1. Cette étape de prétraitement est couramment utilisée dans les modèles de Deep Learning pour normaliser les données d'entrée et les rendre plus facilement interprétables par le réseau neuronal.

Nous redéfinissons également la taille des images, ainsi que l'hyperparamètre 'batch_size' qui permet de définir le nombre d'échantillons qui seront propagés à travers le réseau de neurones en même temps lors de l'entraînement. Plus le batch_size est grand, plus la vitesse d'apprentissage et longue et la qualité des résultats augmente.

Définition des hyperparamètres :

```
# Define some hyperparameters
img_height = 128
img_width = 128
batch_size = 32
num_classes = 7
epochs = 50
```

Création et entraînement du modèle :

Création du modèle :

```
# Create the model
model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(img_height, img_width, 3)))
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'))
```

Compilation du modèle :

```
# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Entraînement du modèle :

```
# Train the model
history = model2.fit(
        train_generator,
        epochs=epochs,
        batch_size=batch_size,
        validation_data=validation_generator)
```

```
Epoch 45/50
33/33 [==============================] - 8s 236ms/step - loss: 0.0182 - accuracy: 0.9951 - val_loss: 3.1556 - val_accuracy: 0.5
814
Epoch 46/50
33/33 [==============================] - 8s 232ms/step - loss: 0.0201 - accuracy: 0.9922 - val_loss: 3.2701 - val_accuracy: 0.5
814
Epoch 47/50
33/33 [==============================] - 8s 239ms/step - loss: 0.0147 - accuracy: 0.9942 - val_loss: 3.5961 - val_accuracy: 0.5
581
Epoch 48/50
33/33 [==============================] - 8s 232ms/step - loss: 0.0197 - accuracy: 0.9951 - val_loss: 3.3020 - val_accuracy: 0.5
465
Epoch 49/50
33/33 [==============================] - 8s 237ms/step - loss: 0.0247 - accuracy: 0.9912 - val_loss: 3.2322 - val_accuracy: 0.5
736
Epoch 50/50
33/33 [==============================] - 8s 233ms/step - loss: 0.0101 - accuracy: 0.9981 - val_loss: 3.3398 - val_accuracy: 0.5
698
```

Evaluation du modèle :

```
# Evaluate the model
score = model.evaluate(validation_generator, verbose=0)
print('Validation loss:', score[0])
print('Validation accuracy:', score[1])
```

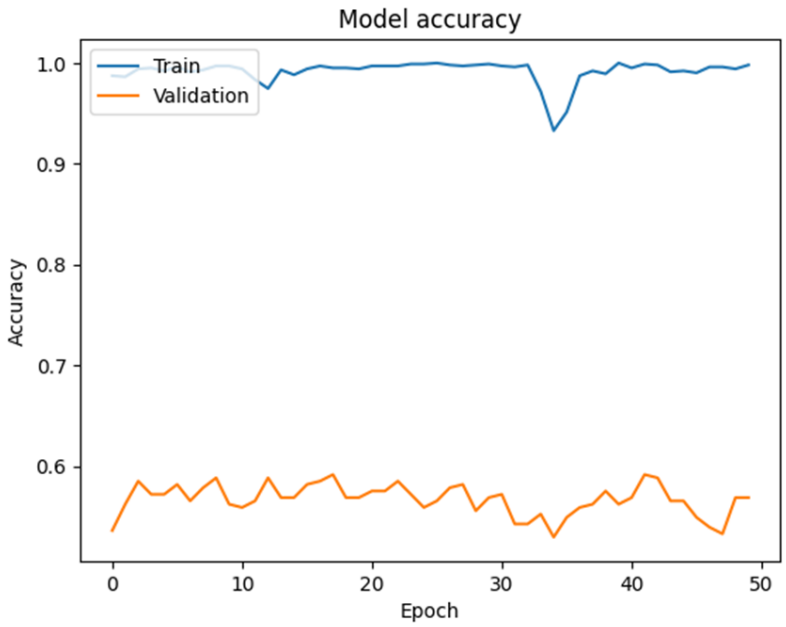
```
Validation loss: 3.3398234844207764
Validation accuracy: 0.569767415523529
```

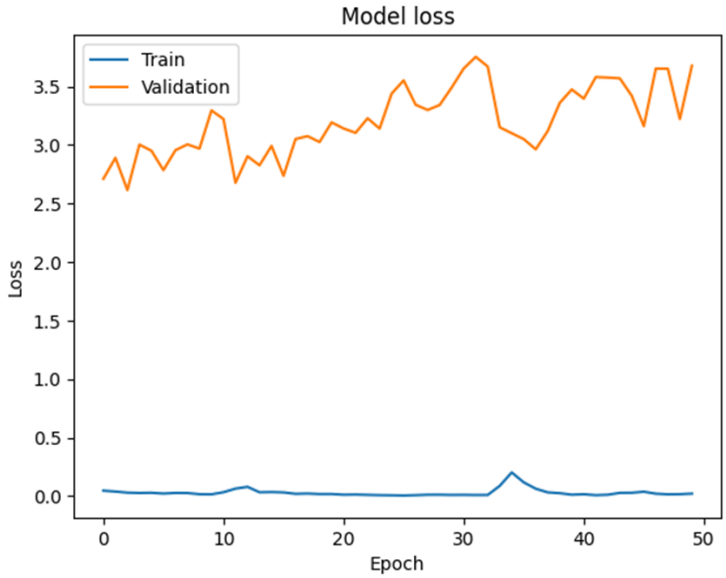
On peut voir que notre modèle à un score de précision de 57% avec une perte de validation de 3.34 ce qui représente l'erreur du modèle sur l'ensemble de validation.

Discussion des résultats :

Tout d'abord la précision de notre modèle ne semble pas être optimale, 57% est un résultat faible surtout avec l'utilisation d'un CNN qui est un des meilleurs modèles pour classifier et faire de la reconnaissance en imagerie.

Nous avons tracé les deux graphiques pour représenter le score de précision et la perte de notre modèle.





D'après les graphiques nous pouvons supposer que le modèle overfit ce qui signifie que le modèle s'est trop adapté aux données d'entraînement spécifiques sur lesquelles il a été entraîné, au point de ne plus généraliser correctement aux nouvelles données.

Il y a plusieurs raisons pour lesquels un modèle pour overfit et cela nous donne des axes d'améliorations pour optimiser notre modèle :

- Vérifier la balance entre données d'entraînement, données de validation

-Entraîner le modèle avec plus de données

- Vérifier le modèle, rajouter des couches ou des hyperparamètres qui permettent de limiter l'overfitting

Il est aussi important de noter la durée d'entraînement du modèle, ici 8 secondes par epochs, donc dans notre cas 8 sec * 50 epochs = 400 secondes soit 400 fois plus long que l'algorithme de machine learning que nous avons choisi.

Une durée d'apprentissage plus longue induit également une consommation bien plus importante entre les deux modèles, les réseaux de neurones et les méthodes de Deep Learning sont beaucoup plus consommateur en matière d'énergie et de temps que les algorithmes de Machine Learning habituels. La durée de l'apprentissage dépend également de la machine sur laquelle le code va tourner. La différence entre l'apprentissage sur un google collab et un jupyter en local sur un ordinateur assez puissant est flagrante, 6,5 min en local contre plus de 50 min sur collab. Donc le deeplearning nécessite des machines très puissantes pour réduire la durée d'apprentissage surtout lorsque les modèles sont complexes.

Conclusion

L’exploitation des données du tableau Excel donné, training_data.csv, a permis d’obtenir un modèle d’une prédiction moyenne d’environ 74%, pouvant aller jusqu’à 80% avec le modèle de machine LinearDiscriminantAnalysis.

Le modèle CNN, un algorithme de Deep Learning est l’un des modèles les plus performants pour la prédiction et l’analyse des images.

Les algorithmes de Machine Learning sont souvent plus simples et plus faciles à mettre en œuvre que les réseaux de neurones profonds, ce qui peut permettre des temps d’entrainement plus courts.

Les algorithmes de Deep Learning peuvent modéliser des relations non linéaires complexes entre les caractéristiques de l'image, ce qui peut permettre des performances de reconnaissance d'image plus élevées que les algorithmes de Machine Learning traditionnels.

Cependant, le score obtenu avec le modèle CNN est de 60%. De plus, l’entrainement du modèle est très long, 7 minutes pour 50 epochs. Ceci peut être dû à un manque de connaissances du modèle de notre part. Dans notre cas le résultat n’est pas vraiment optimal, ne connaissant pas les réseaux de neurones convolutifs nous avons essayé de créer un modèle et de l’entraîner pour avoir un comparatif. Il y a énormément de moyen d‘amléiorer ce résultat. Malgré nos faibles résultats, un CNN reste à ce jour un des meilleurs moyens de réaliser de la classification d’image. Nous avons suivi des articles présentant des entraînements avec un modèle CNN plus complexe, la durée d’apprentissage pouvant durée plus d’1h.

Notre prédiction pourrait être améliorée avec un traitement des données plus performant. Nous pourrons également réaliser au préalable une sélection de features. En effet en ne choisissant que les caractéristiques (points du visage) importantes de notre jeu de données, nous pourrions améliorer notre précision.

Nous pouvons en déduire que si le but est la recherche d’un modèle avec un score de reconnaissance optimal alors il faudra plutôt se tourner vers le deeplearning et l’utilisation d’un CNN. S'il faut réfléchir à une balance entre résultat et durée d’aprentissage/consommation de ressources alors les modèles de machine learning semblent être plus adapté, beaucoup plus rapide et moins coûteux en énergie, ils permettent d’obtenir de bons résultats et présentent une très bonne alternative.