Canran CHEN – Clément GUILLAUME 2019 - 2020

RAPPORT FINAL DU PROJET D'APPLICATION

Sujet 20 : "L'intelligence artificielle au service des Otakus"

kaggle

Remerciements

Avant toute chose, nous tenons à remercier toutes les personnes qui ont permis de mener à bien ce projet.

Nous tenons à remercier en particulier Monsieur Tony RIBEIRO d'avoir su nous guider tout au long du projet. Les différents contenus pédagogiques qu'il nous a procurés, les aides qu'il nous a apportées et sa passion des anime japonais nous ont véritablement permis d'utiliser de nombreuses techniques de *machine learning* et d'en apprendre plus sur l'analyse des données, si fondamentale dans la science des données. Monsieur RIBEIRO a su identifier des problèmes techniques sur Kaggle qui nous empêchaient d'avancer correctement et avec efficacité, nous lui en sommes très reconnaissants. Sans oublier la petite sœur de Monsieur RIBEIRO qui a pertinemment su identifier des biais au sein de nos données.

Nous voulons également remercier Monsieur Morgan MAGNIN de nous avoir suivis tout au long du projet, d'avoir été notre "client" afin d'apporter à ce projet une dimension plus concrète du monde de l'entreprise. Monsieur MAGNIN a fait preuve de bienveillance en mettant en place des réunions fréquentes afin d'avoir un suivi des avancées du projet, nous l'en remercions.

Introduction

La science des données a pour objectif l'extraction de connaissances à partir d'ensembles de données. Elle emploie des techniques et des théories tirées de différents domaines issus des mathématiques et de l'informatique, tels que les statistiques, la programmation, la reconnaissance de formes, la visualisation, le stockage de données, la compression de données et le calcul à haute performance.
Le *Deep Learning* est un sous-ensemble de l'apprentissage automatique (en anglais : *machine learning*) tentant de modéliser avec un haut niveau d'abstraction des données grâce à des techniques basées sur les réseaux de neurones. Depuis une dizaine d'années, ces techniques ont permis des progrès importants et rapides dans les domaines de l'analyse du signal sonore ou visuel et notamment de la reconnaissance faciale, de la reconnaissance vocale, de la vision par ordinateur et du traitement automatisé du langage.

Ce projet consiste en l'utilisation et l'appropriation des différentes techniques de *machine learning* permettant de résoudre des problématiques spécifiques à partir de différents jeux de données tirées de l'animation japonaise.

Les ressources utilisées ont été principalement la plateforme Kaggle offrant des ressources pédagogiques pour les étudiants et permettant une meilleure collaboration entre les étudiants et les encadrants concernant le code réalisé. Nous avons utilisé le langage Python parce que de nombreuses librairies de ce langage sont déjà spécialisées dans les modèles de *machine learning* comme la reconnaissance faciale. Ces librairies sont Scikit-learn, Numpy, Pandas, Shutil, XGBoost et FastAI.

La communication dans ce projet s'est surtout concrétisée au travers de Skype pour les réunion physique parce que l'un des encadrants n'est pas présent physiquement sur le site de Centrale Nantes et au travers de Discord pour s'envoyer des messages, questions et remarques. De manière plus formelle, chaque semaine a été envoyé un rapport d'avancement à chaque protagoniste de ce projet.

Notre projet s'est intéressé principalement à deux *datasets* distincts. Le premier *dataset* est un tableau comportant une trentaine de *features* en colonne et 14 000 entrées tabulaires. Le deuxième *dataset* est constitué de plus de 300 000 images d'anime japonais.

SOMMAIRE

I- ANALYSE EXPLORATOIRE DES DONNEES

L'exploratory data analysis est une étape d'analyse de données permettant d'identifier les principales caractéristiques d'un *dataset*. Dans le cadre de ce projet, nous avons utilisé deux types de données : des données tabulaires et des images.

1.1. Importer les librairies

Afin de réaliser les analyses préalables du *dataset*, nous avons importé les librairies comportant les fonctions nécessaires à cette analyse. Les librairies importées sont les packages *Numpy* et *Pandas*.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

Numpy est la librairie de référence en Python pour manipuler des tableaux ou matrices multidimensionnels et leur appliquer des fonctions mathématiques, notamment des calculs numériques et des fonctions d'algèbre linéaire. C'est pourquoi la librairie *Numpy* a été utilisée à la fois pour les données tabulaires mais aussi pour les images, étant considérées comme des tableaux de pixels lors des manipulations d'images.

La librairie *Pandas* sert plutôt à manipuler des données grâce à l'objet DataFrame présent dans sa librairie. Le *data processing* permis par cette librairie nous a permis d'obtenir des informations plus détaillées du *dataset*. Nous avons aussi utilisé *Pandas* pour importer les fichiers CSV qui constituaient notre *dataset*.

1.2. Données tabulaires

1.2.1. Exploration et Préparation des données

Pour analyser les données tabulaires, nous avons utilisé les fonctions *info()* et *head()* qui permettent d'afficher la structure et l'aperçu de chaque dataframe.

La fonction *info()* retourne le nombre total d'entrées du tableau en précisant l'intervalle des indices, elle retourne aussi le nombre total de colonnes puis pour chaque colonne, son nom, le nombre d'occurrences non vides et le type des variables présentes dans la colonne. Elle compte le nombre de types primaires rencontrés comme le nombre de booléens, de réels, d'entiers et d'objets par exemple et elle finit par donner le coût de stockage en mémoire du *dataset*.

La fonction *head()* affiche les 5 premières entrées du *dataset* colonne par colonne et les dimensions du tableau ainsi affiché.

Lors de la data exploration, on peut obtenir des résultats plus détaillés avec la librairie Pandas en réalisant un *profiling report*.

```
import pandas_profiling as pp
report = pp.ProfileReport(anime_filtered_data)
report.to_file("report.html")

report
```

Ce *profiling report* analyse chaque variable et si la variable est quantitative, le *profiling report* spécifie le nombre d'entrées différentes en absolu et en pourcentage du nombre total des entiers, le pourcentage d'entrées manquantes, d'entrées infinies, la moyenne, le minimum, le maximum et le pourcentage de zéros. Si la variable est qualitative, nous obtenons uniquement le nombre d'entrées différentes en absolu et en pourcentage du nombre total des entiers, le pourcentage d'entrées manquantes.

Si un résultat du rapport paraît anormal, il est écrit en rouge.

Dataset info

Number of variables	31
Number of observations	14474
Missing cells	73271 (16.3%)
Duplicate rows	0 (0.0%)
Total size in memory	3.3 MiB
Average record size in memory	241.0 B

Variables types

Numeric	7
Categorical	21
Boolean	1
Date	0
URL	1
Text (Unique)	0
Rejected	1
Unsupported	0

Warnings

`aired` has a high cardinality: 9649 distinct values	Warning
`aired_string` has a high cardinality: 10026 distinct values	Warning
`background` has a high cardinality: 1039 distinct values	Warning
`background` has 13417 (92.7%) missing values	Missing
`broadcast` has a high cardinality: 442 distinct values	Warning
`broadcast` has 10203 (70.5%) missing values	Missing
`duration` has a high cardinality: 301 distinct values	Warning
`ending_theme` has a high cardinality: 5458 distinct values	Warning
`episodes` is highly skewed (γ1 = 24.51511134)	Skewed

duration
Categorical

Distinct count	301
Unique (%)	2.1%
Missing (%)	0.0%
Missing (n)	0

24 min. per ep.	1513
25 min. per ep.	808
23 min. per ep.	797
Other values (298)	11356

Toggle details

ending_theme
Categorical

Distinct count	5458
Unique (%)	37.7%
Missing (%)	0.0%
Missing (n)	0

[]	8803
['"Reckless" by LAZY']	6
['"SERIOUS-AGE" by Faylan']	5
Other values (5455)	5660

Toggle details

episodes
Numeric

Distinct count	196
Unique (%)	1.4%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0

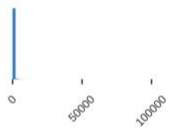
Mean	11.3109714
Minimum	0
Maximum	1818
Zeros (%)	3.5%

Toggle details

favorites
Numeric

Distinct count	1229
Unique (%)	8.5%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0

Mean	311.7355949
Minimum	0
Maximum	106895
Zeros (%)	34.6%



Toggle details

genre
Categorical

Distinct count	4545
Unique (%)	31.4%
Missing (%)	0.4%
Missing (n)	64

Hentai	867
Comedy	590
Music	533
Other values (4541)	12420

1.2.2. Visualisation des données

Concernant les données tabulaires, nous nous sommes intéressés à des *features* spécifiques qui portaient le plus de sens dans le tableau, et des *features* que l'on pouvait croiser avec bon sens, de préférence quantitatives pour obtenir des statistiques plus détaillées. On a ainsi affiché feature par feature les extremas de ces variables quantitatives pour analyser la répartition des données en supprimant les entrées comprenant des valeurs manquantes (*NaN*) en utilisant la fonction *dropna()*.

Les variables que nous avons analysées sont:

- Duration (sous format texte, mais d'unité constante, juste tokenizer le texte pour en extraire les heures, minutes et secondes)
- Episodes (entier)
- Genre (format texte)
- Source (format texte)
- Type (format texte)
- Score (entier)
- Tags (format texte)

Certaines de ces variables sont qualitatives, comme les variables Duration, Genre, Source, Type et Tags. Pour pouvoir exploiter dans un modèle de *machine learning* les variables qualitatives, nous avons réalisé un one-hot-encoding, c'est-à-dire nous avons affecté à chaque entrée qualitative un nombre distinct deux à deux afin de les identifier de manière quantitative. Il en résulte un vecteur binaire.

Pour la feature Source, nous avons réalisé un one-hot-encoding en utilisant la fonction *LabelBinarizer()* qui nous a permis d'identifier et de distinguer comme expliqué les différentes sources. Il en résulte un tableau qui prend en colonnes toutes les sources possibles et en lignes les différentes entrées du *dataset*. Comme plusieurs sources étaient possibles par entrée, on ne pouvait pas réaliser un encodage unique pour chaque source. Nous avons ainsi réalisé une matrice qui permettait de prendre en considération tous les cas possibles.

Nous avons également utilisé la fonction *describe()* de la librairie Pandas afin d'obtenir des indicateurs statistiques fréquemment utilisés pour analyser chaque feature quantitative comme la moyenne, les quartiles, les valeurs minimum et maximum.

```
X.describe()
```

:

	episodes
count	14474.000000
mean	11.310971
std	43.449161
min	0.000000
25%	1.000000
50%	1.000000
75%	12.000000
max	1818.000000

1.3. IMAGES

1.3.1. Hyperparamètres

En *machine learning*, nous pouvons utiliser des hyperparamètres, paramètres définis avant l'apprentissage automatique. Ces hyperparamètres concernent en général la taille et le type de neurone dans un réseau de neurones.

Dans ce projet, nous avons établi un hyperparamètre qui fixait le nombre d'images analysées parce que la plateforme de développement Kaggle prenait trop de temps concernant le traitement d'un grand nombre d'images. Nous avons donc fixé au départ cet hyperparamètre à 1000. Nous avons choisi les 1000 premières entrées du *dataset* sur les plus de 2 millions d'images à disposition.

```
samples = 1000
```

1.3.2. Chargement des données

Nous avons chargé les données sur la plateforme Kaggle au moyen de la fonction *read_csv()* qui prend en paramètres l'adresse du fichier csv et le nombre d'entrées à charger. Pour vérifier le bon chargement, nous utilisons les fonctions *info()* et *head()* afin d'avoir le nombre total des entrées chargées dans le Dataframe.

```
df_raw = pd.read_csv('/kaggle/input/safebooru/all_data.csv', nrows = samples)
df_raw.head(samples)
```

id	created_at	rating	score	sample_url	sample_width	sample_height	preview_
1	1264803292	s	37	//safebooru.org/samples/1/sample_e7b3dc281d431...	850	638	//safebo
2	1264803292	s	12	//safebooru.org/samples/1/sample_27ff11b17a2c3...	850	1208	//safebo
3	1264803298	s	8	//safebooru.org/samples/1/sample_ebd16eb1d1547...	850	599	//safebo
4	1264803299	s	5	//safebooru.org/samples/1/sample_6fbb9a4b9099e...	850	519	//safebo
6	1264803304	s	11	//safebooru.org/samples/1/sample_113cb63dc5412...	850	601	//safebo
...	...	...	...	...	...	...	...
1174	1264804365	s	3	//safebooru.org/samples/2/sample_f34553d206f48...	850	638	//safebo

```
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
id              1000 non-null int64
created_at      1000 non-null int64
rating          1000 non-null object
score           1000 non-null int64
sample_url      1000 non-null object
sample_width    1000 non-null int64
sample_height   1000 non-null int64
preview_url     1000 non-null object
tags            1000 non-null object
dtypes: int64(5), object(4)
memory usage: 70.4+ KB
```

1.3.3. Ingénierie des données

Une fois que nous avons chargé les fichiers csv et les entrées, il faut sélectionner les colonnes que l'on va analyser. On s'intéresse aux colonnes "sample_url" et "tags". Pour ce faire, nous avons créé un nouveau tableau qui est dans les faits le sous-tableau des colonnes qui nous intéressent.

```
features = ["sample_url", "tags"]
df_X = df_raw[features]
df_X.columns
```

```
Index(['sample_url', 'tags'], dtype='object')
```

On réalise ensuite l'extraction des tags. En effet, la colonne "tags" du *dataset* que nous avons analysé correspondante aux différents tags qui décrivent l'image associée. Comme l'objectif de notre modèle a été d'identifier le genre du personnage de l'image, nous avons donc isolé les tags correspondant à notre objectif, c'est-à-dire: 1girl, 1boy,... en utilisant un tokenizer qui enregistre ensuite dans un dictionnaire l'ensemble des clefs (ici les tags) et leurs valeurs (leur nombre). Pour visualiser le résultat, on peut par exemple en afficher 5.

```
import re
tag = []

for i in df_X.tags:
    tokens = re.split("[ ]",i)
    for token in tokens:
        if token not in tag:
            tag.append(token)
print("There are", len(tag), "different tags")

tag[:10]
```

```
chose = []
dic = {}
for i in df_X.tags:
    tokens = re.split("[ ]",i)
    for token in tokens:
        if token in ['1girl', 'bag','black_hair','blush','bob_cut']:
            chose.append(token)
for j in chose:
     dic[j] = dic.get(j,0)+1
print("5 tags occurences:",dic)
```

```
5 tags occurences: {'1girl': 79, 'bag': 5, 'black_hair': 76, 'blush': 188, 'bob_cut': 8}
```

On obtient ainsi l'ensemble des tags et leur nombre dans le *dataset*. Il faut vérifier que l'on a *solo* qui semble identifier des personnage seuls. On observe aussi *1girl* qui peut indiquer une seule fille sur l'image, on peut s'attendre à avoir un tag *1boy*. On a également le tag *male* qui identifie la présence de personnage masculin.

```
if EDA:
    print("Occurences: ")
    selected_tags = ["solo", "1girl", "1boy", "girl", "female", "loli", "boy", "male", "ma
n", "shota", "trap", "reverse_trap"]
    for tag in selected_tags:
        if tag in tags:
            print(tag, tags[tag])
        else:
            print(tag," ",0)
```

```
Occurences:
solo 227
1girl 79
1boy 20
girl 1
female 5
loli 17
boy 1
male 59
man   0
shota   0
trap 26
reverse_trap   0
```

Les tags retenus pour identifier des personnages féminin ou masculin seul seront :

- tags 1 personnage : solo
- tags féminins : 1girl, girl, female, loli
- tags masculins : 1boy, boy, male, man, shota
- tags trap : trap, reverse_trap

On crée une nouvelle colonne “target” qui encode les tags décrivant les garçons avec un 2 et les filles avec un 1. On obtient ainsi un tableau comportant 92 entrées. On charge ensuite les images à partir de leurs adresses url. On supprime les adresses url et on les remplace par leurs adresses de dossier sur Kaggle.

Afin de réutiliser les données tabulaires ainsi traitées, on en crée un zip et un csv afin de les réutiliser dans d’autres kernels dédiés à l’apprentissage automatique. Pour ce faire, nous avons importé la librairie *Shutil* et nous avons utilisé la fonction *make_archive* avec en paramètres le titre, le format et l’endroit de stockage sur le kernel du fichier à compresser.

```
df_X.to_csv("sanboru_gender_dataset.csv", index=False)
```

```
import shutil
shutil.make_archive("sanboru_gender_dataset", 'zip', "train")
!rm train/*
!rm -d "train"
```

```
!ls
```

```
__notebook__.ipynb  sanboru_gender_dataset.csv
__output__.json     sanboru_gender_dataset.zip
```

1.3.4. Limites des données

Au cours de notre projet, nous avons pu constater quelques limites du *dataset*. En effet, certaines données peuvent être source d’erreur. Par exemple, une image comportant un garçon seul qui ne pouvait pas être identifié à l’oeil humain comme tel. Ces personnages avaient une apparence androgyne qui pouvait tout aussi bien tromper le modèle d’apprentissage automatique. Certaines images de notre *dataset* correspondaient à ce type de donnée et introduisaient des erreurs d’apprentissage.

D’autres sources d’erreurs pouvaient provenir de la répartition statistique des données. En effet, le *dataset* que l’on avait obtenu à partir des 1000 premières entrées du *dataset* original était constitué de 13 images de garçon et 80 images de fille… Cette répartition très déséquilibrée entraînait un biais d’apprentissage très fort pour nos modèles et ne permettait pas d’obtenir un modèle avec des performances satisfaisantes sur un autre *dataset*.

De même pour le *dataset* tabulaire, nous avons remarqué qu’un certain nombre d’animes ont un score de 0. En analysant plus en détails, on a remarqué que les dates de sortie de ces anime étaient à venir. On en a déduit que la valeur 0 dans la colonne des scores a été la valeur mise par défaut quand il n’y en avait pas. Cela a introduit un biais dans l’analyse des scores parce que l’on prenait en compte un anime alors qu’il n’était pas encore diffusé…

En conclusion, les différents types de données que nous avons analysés nous ont permis d'apprendre les techniques d'analyse et de traitement leur correspondantes. Nous avons également constaté l'importance de l'étude préalable des données et les conséquences d'un *dataset* pas harmonisé sur le modèle d'apprentissage automatique.

II- MACHINE LEARNING (Données Tabulaires)

Le *machine learning* est une sous-catégorie de l'intelligence artificielle. Il peut être vu comme l'implémentation d'algorithmes apprenant à partir de données en utilisant des outils statistiques. Ces données vont permettre à l'algorithme d'établir certaines propriétés à partir de ces données et ensuite de pouvoir résoudre différents problèmes, essentiellement de classification à partir d'autres données.

Les algorithmes de *machine learning* sont souvent divisés en deux catégories : les algorithmes d'apprentissage supervisé lorsque les classes d'appartenance des données sont connues à l'avance, et d'apprentissage non supervisé lorsque ni leur nombre ni leur nature ne sont connus et vont être déterminés par l'algorithme lui-même. Notre projet s'intéresse uniquement à l'apprentissage supervisé.

Une fois le travail sur les données effectué, nous avons prédit le score des animations japonaises de l'échantillon de test à l'aide de différents algorithmes présents dans les bibliothèques *Scikit-learn, XGboost et Keras* de Python. Le détail du fonctionnement de ces algorithmes est expliqué ci-dessous.

2.1. Arbre de décision - Decision tree

2.1.1. Principe de l'arbre de décision

Dans ce modèle de *machine learning*, toutes les variables doivent être catégorielles. Il faut donc créer plusieurs catégories pour les variables quantitatives.

L'arbre de décision est un arbre au sens algorithmique avec des nœuds et des branches. Partant d'un noeud, chaque branche correspond au choix d'une catégorie pour une certaine variable. Chaque feuille de l'arbre correspond à une situation pour laquelle toutes les catégories de chaque variable ont été déterminées et indique la prédiction correspondante. L'algorithme consiste donc, étant donnée une entrée, à se balader dans l'arbre depuis la racine en considérant les catégories d'appartenance aux variables de cette entrée jusqu'à arriver à une feuille de l'arbre indiquant alors la prédiction pour cette entrée.

2.1.2. Arbre de décision dans la pratique

Il faut d'abord importer *DecisionTreeRegressor* à partir de la librairie *Scikit-learn.* Et apràs on définit le modèle et on l'ajuste.

```
from sklearn.tree import DecisionTreeRegressor
```

```
# Define model
model = DecisionTreeRegressor()
# Fit model
model.fit(train_X, train_y)
```

2.1.3. Evaluation

L'évaluation le plus basique de l'arbre de décision est *mean_absolute_error*.

```
val_predictions = model.predict(val_X)
MAE_DT = mean_absolute_error(val_y, val_predictions)
print("Model MAE: ", mean_absolute_error(val_y, val_predictions))
```

```
Model MAE:  0.8084328191515925
```

Pour savoir si la MAE est un bon révélateur des performances du modèle, on calcule la différence entre les valeurs origines et les valeurs en prédictions.

```
results = pd.DataFrame({'Original': val_y, 'Predictions':val_predictions, 'difference': [abs
(x1 - x2) for (x1, x2) in zip(val_y, val_predictions)]})
results
```

	Original	Predictions	difference
9058	7.39	6.73	0.66
471	7.26	7.14	0.12
14321	5.29	5.00	0.29
4368	5.42	5.15	0.27
9493	7.86	8.35	0.49
...	...	...	...
11511	6.72	7.04	0.32
8892	6.26	7.49	1.23
13654	5.83	6.33	0.50
4943	8.08	6.71	1.37
2813	5.75	5.45	0.30

1974 rows × 3 columns

De plus, on pourrait discrétiser nos scores en mauvais/moyen/bon par exemple et évaluer le modèle sur ce range. Ici on définit une fonction *discrete_precision* pour réaliser cette idée et on l'appliquera à tous les modèles suivants pour évaluer la fidélité de la prédiction.

```
accuracy_DT = discrete_precision(val_y, val_predictions, DISCRETE_VALUES)
print("Dans", accuracy_DT , "% des cas le modèle prédit le bon range sur",DISCRETE_VALUES,"r
anges.")
```

```
Dans 69.35 % des cas le modèle prédit le bon range sur 4 ranges.
```

2.2. Forêt d'arbres décisionnels - Random Forest

2.2.1. Principe de la forêt d'arbres décisionnels

Ce modèle est une version améliorée de celle des arbres de décision. Il effectue un apprentissage à partir de plusieurs arbres de décision entraînés sur des sous-ensembles de données.

Nous avons réalisé un random forest en utilisant la fonction *RandomForestRegressor()* de la librairie *Sklearn*. Nous avons laissé le nombre d'arbres de décision par défaut, égal à 100 et nous avons renseigné le paramètre *random_state* égal à 0 ce qui nous a permis d'obtenir un modèle qui effectue à chaque exécution le même traitement des données.

2.2.2. Forêt d'arbres décisionnels dans la pratique

On importe *RandomForestRegressor()* de la librairie *Scikit-learn* et la fonction *mean_absolute_error* de *Scikit-learn* afin de mesurer les performances du modèle.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

model = RandomForestRegressor(random_state=1)
model.fit(train_X, train_y)
```

2.2.3. Evaluation

Nous avons mesuré l'efficacité du modèle avec la *mean_absolute_error* et aussi en utilisant la fonction *discrete_precision*.

```
val_predictions = model.predict(val_X)
MAE_RF = mean_absolute_error(val_y, val_predictions)
print("Model MAE: ", mean_absolute_error(val_y, val_predictions))
accuracy_RF = discrete_precision(val_y, val_predictions, DISCRETE_VALUES)
print("Model accuracy: ", accuracy_RF, "% sur",DISCRETE_VALUES,"ranges.")
```

```
Model MAE:  0.6455489806850438
Model accuracy:  74.77 % sur 4 ranges.
```

2.3. XGBoost

2.3.1. Principe du XGBoost

Cette technique de boosting est majoritairement employée avec des arbres de décision. L'idée principale est qu'au lieu d'utiliser un seul modèle, l'algorithme va en utiliser plusieurs qui seront ensuite combinés pour obtenir un seul résultat. C'est avant tout une approche pragmatique qui permet donc de gérer des problèmes de régression comme de classification.

Pour décrire succinctement le principe, l'algorithme travaille de manière séquentielle. Contrairement par exemple au modèle Random Forest, ce procédé rend l'exécution de l'algorithme plus lente mais cela va permettre à notre algorithme de s'améliorer par

capitalisation par rapport aux exécutions précédentes. Il commence donc par construire un premier modèle qu'il va bien sûr évaluer. On conserve par la suite le meilleur modèle pour les prédictions.

2.3.2. XGBoost par la pratique

XGBoost ne fait pas partie de Scikit-Learn mais s'intègre parfaitement bien avec. Les algorithmes ci-dessous utilisent cette intégration, mais sachez que nous pouvons utiliser XGBoost sans avoir Scikit-Learn.

Vu que l'on étudie un problème de régression, nous utiliserons donc XGBoost comme regressor.

```
from xgboost import XGBRegressor
import xgboost as xgb

model = XGBRegressor(n_estimators=1000, learning_rate=0.05, n_jobs=4)
model.fit(train_X, train_y,
             early_stopping_rounds=5,
             eval_set=[(val_X, val_y)],
             verbose=True)

val_predictions = model.predict(val_X)
MAE_XGB = mean_absolute_error(val_y, val_predictions)
print("Model MAE: ", mean_absolute_error(val_y, val_predictions))
accuracy_XGB = discrete_precision(val_y, val_predictions, DISCRETE_VALUES)
print("Model accuracy: ", accuracy_XGB, "% sur",DISCRETE_VALUES,"ranges.")
```

2.3.3. Evaluation

De même qu'avec le random forest, on utilise la mean absolute error pour analyser l'efficacité d'apprentissage du modèle.

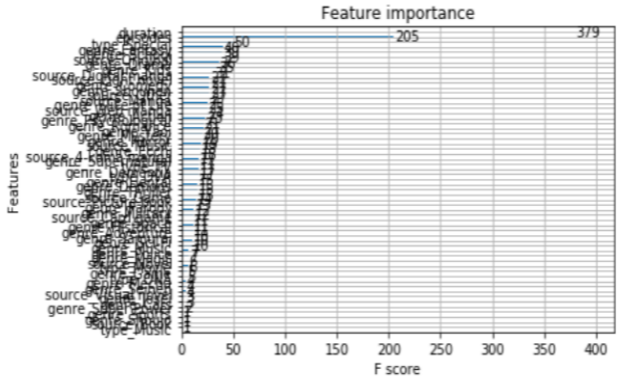
```
Model MAE:  0.6431761682214708
Model accuracy:  75.84 % sur 4 ranges.
```

En outre, la librairie XGBoost fournit quelques métriques intéressantes.

Graphique présentant les champs par degré d'importance:

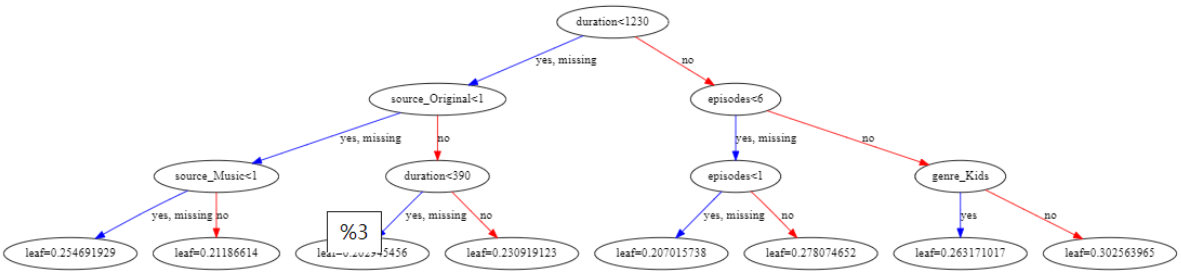
```
xgb.plot_importance(model)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f068a331908>
```



Graphique présentant le résultat:

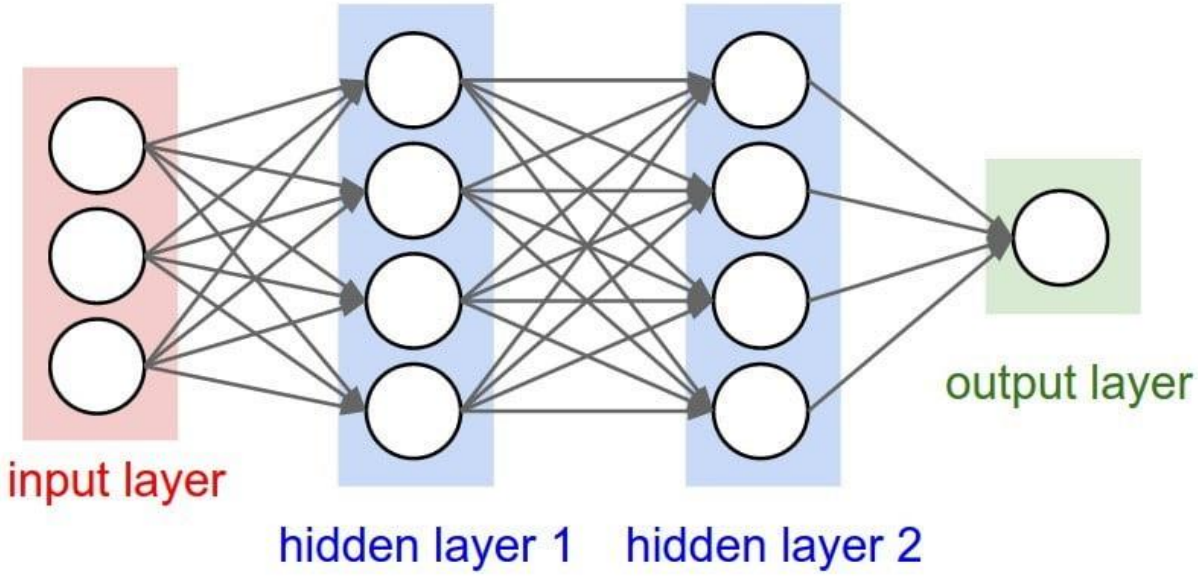
```
xgb.to_graphviz(model, num_trees=2)
```



2.4. Réseau de neurones - Neural Network

2.4.1. Principe du réseau de neurones

Un réseau de neurone artificiels ou Neural Network s'inspire du fonctionnement du cerveau humain pour apprendre.

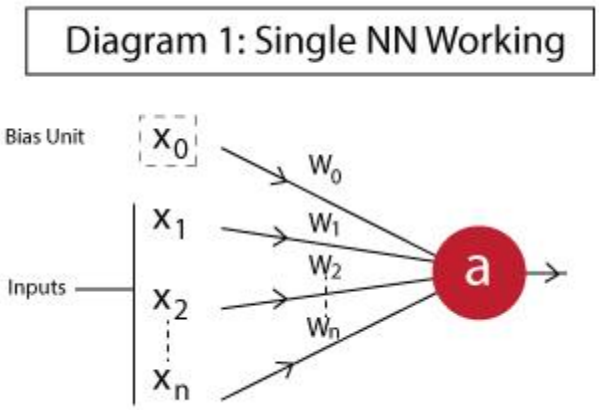


En règle générale, un réseau de neurones repose sur un grand nombre de processeurs opérant en parallèle et organisés en tiers.

Il est divisé en couche de 3 types :

1. Couche d'entrée : les observations d'entraînement sont alimentées par ces neurones
2. Couches cachées : ce sont les couches intermédiaires entre l'entrée et la sortie qui aident le réseau neuronal à apprendre les relations compliquées impliquées dans les données.
3. Couche de sortie : la sortie finale est extraite des deux couches précédentes. Par exemple : En cas de problème de classification avec 5 classes, la sortie aura plus tard 5 neurones.

Dans cette partie, nous explorerons le fonctionnement d'un seul neurone. Un neurone typique ressemble à :



Les différents composants sont :

1. x1, x2, …, xN : entrées vers le neurone. Il peut s'agir des observations réelles de la couche d'entrée ou d'une valeur intermédiaire de l'une des couches cachées.

2. x0 : unité de biais. Il s'agit d'une valeur constante ajoutée à l'entrée de la fonction d'activation. Il fonctionne de manière similaire à un terme d'interception et a généralement une valeur +1.
3. w0, w1, w2, …, wN : poids sur chaque entrée. Notez que même l'unité de biais a un poids.
4. a : Sortie du neurone qui est calculée comme :

$$a = f(\sum_{i=0}^{N} w_i x_i)$$

5. Ici f est connue une fonction d'activation. Cela rend un réseau neuronal extrêmement flexible et confère la capacité d'estimer des relations non linéaires complexes dans les données. Il peut s'agir d'une fonction gaussienne, d'une fonction logistique, d'une fonction hyperbolique ou même d'une fonction linéaire dans des cas simples.

2.4.2. Réseau de neurones dans la pratique

A l'aide de la bibliothèque *Keras*, on applique un réseau de neurones à notre *dataset*. Nous avons construit deux modèles, l'un a seulement un neurone et l'autre est constitué de 2 couches cachées, le premier de 16 neurones et le second de 8 neurones.

Les étapes de la création d'un réseau de neurones sont les suivantes :

- **Définir le modèle :** Pour construire un modèle de réseau de neurone, nous devons définir les couches (entrée, caché et sortie). Ici, nous irons de l'avant avec un modèle séquentiel, ce qui signifie que nous définirons les couches de manière séquentielle. De plus, nous irons de l'avant avec un réseau entièrement connecté. **1.** Premièrement, nous nous concentrerons sur la définition de la couche d'entrée. Cela peut être spécifié quand on crée la première couche avec l'argument *input_dim* et la définit comme *train_X.shape[1]* pour les variables indépendantes. **2.** Ensuite, définissez le nombre de couches cachées ainsi que le nombre de neurones et de fonctions d'activation. Le bon nombre peut être atteint en passant par plusieurs itérations. Plus le nombre est élevé, plus votre modèle est complexe. Pour le premier modèle, on ne l'utilise pas et pour l'autre, on en a 2, premier de 16, second de 8. **3.** Enfin, nous devons définir la couche de sortie avec 1 neurone pour prédire le score. Le problème à résoudre est un défi de régression afin que nous puissions aller de

l'avant avec une transformation linéaire au niveau de la couche de sortie. Par conséquent, il n'est pas nécessaire de mentionner une fonction d'activation (elle est linéaire par défaut).

```
# define the keras model
model = Sequential()
model.add(Dense(1, input_dim=train_X.shape[1]))
```

```
# define the keras model
model = Sequential()
model.add(Dense(16, input_dim=train_X.shape[1]))
model.add(Dropout(0.2))
model.add(Dense(8))
model.add(Dropout(0.1))
model.add(Dense(1))
```

Remarque : Ici, pour définir le deuxième modèle, on a appliqué une technique de prévention de surentraînement *DropOut.* L'idée de cette technique est d'ignorer certains neurones de manière aléatoire dans notre réseau pour que le modèle ne se concentre pas sur un petit nombre de patterns spécifiques au *train dataset* et qu'il puisse trouver de nouveaux moyens de résoudre un même problème avec de nouveaux patterns. Cette technique de Dropout est utilisée lorsque l'on observe une moindre performance dans le *validation dataset* que dans le *train dataset* pour des neurones qui obtiennent d'importantes performances.

- **Compiler le modèle :** Dans cette étape, nous allons configurer le modèle pour la "train". Nous allons configurer l'optimiseur pour changer les poids et les biais, et la fonction de perte et la métrique pour évaluer les performances du modèle. Ici, nous utiliserons "*rmsprop*" comme optimiseur, "*mean squared error*" comme métrique de perte. Selon le type de problème que nous résolvons, nous pouvons modifier nos pertes et nos mesures.

```
# compile the keras model
model.compile(optimizer='rmsprop',
              loss='mse')
```

- **Ajuster le modèle :** Maintenant, la dernière étape de la construction du modèle consiste à ajuster le modèle sur le trainset (qui représente en fait 70% de l'ensemble de données complet). Nous devons fournir des variables indépendantes et dépendantes ainsi que le nombre d'itérations d'entraînement, c'est-à-dire les époques. Ici, nous avons pris 150 époques.

```
# fit the keras model on the dataset
model.fit(train_X, train_y, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=1, validation_data
=(val_X,val_y))
```

2.4.3. Evaluation

Les algorithmes qu'on applique ici sont similaires. Vu que *mean_absolute_error* du deuxième modèle est plus petit et *accuracy* est plus grand, les performances de notre deuxième modèle sont mieux que celles du premier modèle.

```
val_predictions = model.predict(val_X)
MAE_LR = mean_absolute_error(val_y, val_predictions)
print("Model MAE: ", mean_absolute_error(val_y, val_predictions))
accuracy_LR = discrete_precision(val_y, val_predictions, DISCRETE_VALUES)
print("Model accuracy: ", accuracy_LR, "% sur",DISCRETE_VALUES,"ranges.")
```

```
Model MAE:  1.3345017676730286
Model accuracy:  62.01 % sur 4 ranges.
```

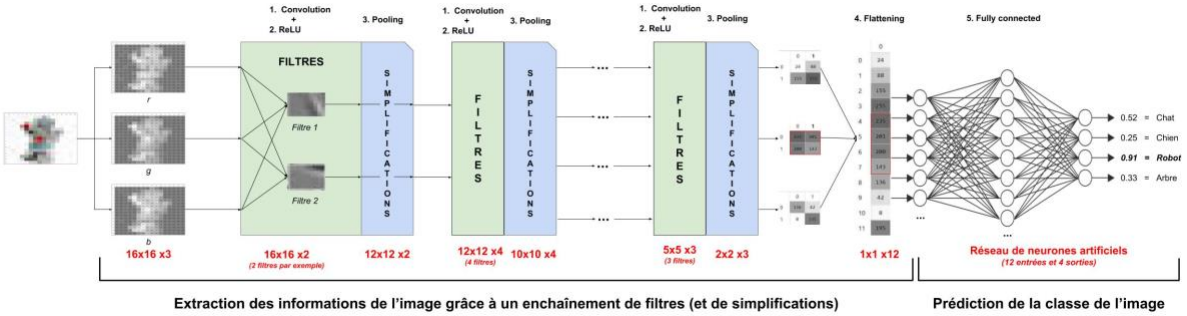
```
val_predictions = model.predict(val_X)
MAE_NN = mean_absolute_error(val_y, val_predictions)
print("Model MAE: ", mean_absolute_error(val_y, val_predictions))
accuracy_NN = discrete_precision(val_y, val_predictions, DISCRETE_VALUES)
print("Model accuracy: ", accuracy_NN, "% sur",DISCRETE_VALUES,"ranges.")
```

```
Model MAE:  0.6979982067676301
Model accuracy:  73.2 % sur 4 ranges.
```

2.5. Réseau de neurones profonds - Deep Neural Network

2.5.1. Principe du réseau de neurones profonds

Les réseaux de neurones profonds peuvent comporter des millions de neurones, répartis en plusieurs dizaines de couches. L'espoir est que plus on augmente le nombre de couches, plus les réseaux de neurones apprennent des choses compliquées, abstraites, correspondant de plus en plus à la manière dont un humain raisonne. Il est néanmoins difficile de mettre au point des mécanismes d'apprentissage efficaces pour chacune des couches intermédiaires (appelées couches profondes ou couches cachées).



2.5.2. Réseau de neurones profonds dans la pratique

On a défini 7 couches cachées de 128 à 2 neurones, décrémentant en puissance de 2 (128,64, etc.). On a aussi appliqué la technique *DropOut.*

```
# define the keras model
model = Sequential()
model.add(Dense(128, input_dim=train_X.shape[1]))
model.add(Dropout(0.4))
model.add(Dense(64))
model.add(Dropout(0.3))
model.add(Dense(32))
model.add(Dropout(0.15))
model.add(Dense(16))
model.add(Dropout(0.1))
model.add(Dense(8))
model.add(Dropout(0.05))
model.add(Dense(4))
model.add(Dropout(0.015))
model.add(Dense(2))
model.add(Dense(1))
# compile the keras model
model.compile(optimizer='rmsprop',
              loss='mse')
# fit the keras model on the dataset
model.fit(train_X, train_y, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=1, validation_data
=(val_X,val_y))
```

2.5.3. Evaluation

Comparées à celles du précédent modèle, les performances de ce modèle n'ont pas beaucoup été améliorées.

```
val_predictions = model.predict(val_X)
MAE_DL = mean_absolute_error(val_y, val_predictions)
print("Model MAE: ", mean_absolute_error(val_y, val_predictions))
accuracy_DL = discrete_precision(val_y, val_predictions, DISCRETE_VALUES)
print("Model accuracy: ", accuracy_DL, "% sur",DISCRETE_VALUES,"ranges.")
```

```
Model MAE:  0.697279040279601
Model accuracy:  73.25 % sur 4 ranges.
```

2.6. Comparaison des modèles et leurs limites

Pour chaque modèle, on évalue sa performance en mean absolute error (MAE) qui existe déjà dans la librairie *Scikit-learn*. De plus, nous avons nous-même défini une fonction *discrete_precision* qui discrétise nos scores en mauvais/moyen /bon et évalue le modèle sur ce range.

```
pd.DataFrame(np.array([[MAE_DT,MAE_RF,MAE_XGB,MAE_LR,MAE_NN,MAE_DL],
                       [accuracy_DT,accuracy_RF,accuracy_XGB,accuracy_LR, accuracy_NN,accuracy_DL]]),
             columns=['decision Tree', 'Random Forest', 'XGBoost', 'Logistic Regression', 'Neural Network', 'Deep Learning'],
            index = ["MAE","accuracy"])
```

:

	decision Tree	Random Forest	XGBoost	Logistic Regression	Neural Network	Deep Learning
MAE	0.808433	0.645549	0.643176	1.334502	0.697998	0.697279
accuracy	69.350000	74.770000	75.840000	62.010000	73.200000	73.250000

L'idée de cette partie est de savoir comment nous pouvons construire les modèles différents sur un ensemble de données structure afin que nous ne nous concentrions pas sur d'autres aspects de l'amélioration des prédictions du modèle. Mais il n'est pas difficile de trouver que le modèle XGBoost fait preuve d'une performance impressionnante.

III- Analyse des données image par Deep Learning

Après avoir utilisé les différents modèles présentés précédemment, nous avons utilisé le modèle de Deep Learning pour analyse le *dataset* d'images parce que ce modèle est plus adapté à l'analyse des images parmi les autres modèles. Voici comment nous avons procédé afin d'entraîner un modèle de Deep Learning d'identification du genre des personnages d'anime japonais.

3.1. Data Architecture

Au niveau des données, nous avons repris les images que nous avons identifiées dans le *dataset* d'origine et nous les avons réparties dans différents dossiers afin de séparer le train set et le validation set. Pour ce faire, nous avons importé dans ce kernel l'output du kernel que l'on a créé pour identifier les images, nous avons décompressé le fichier zip et nous avons créé un dossier train divisé en deux dossiers fille et garçon. Nous avons également créé un dossier validation selon le même schéma que précédemment.

Nous avons ensuite affecté chaque image au dossier correspondant selon la colonne target permettant de distinguer les images de fille et de garçon. Cette architecture est nécessaire afin d'utiliser la fonction ResNet50 qui fait appel à ces neurones déjà entraînés pour analyser une image.

3.2. Data préparation

Après avoir fait l'architecture de données des fichiers, nous avons préparé les données. En effet, chaque image que nous avons sélectionnée n'est pas au même format, ce qui ne permet pas d'entraîne directement un modèle d'apprentissage automatique. Il est nécessaire d'établir un format d'image identique au préalable.

Nous avons défini une fonction appelée *read_and_prep()* qui reformate les images du *dataset* dans un même format: 224p sur 224p.

3.3. Modèle utilisant ResNet50

3.3.1. Principe du modèle ResNet50

ResNet50 (Residual Networks 50) est un réseau de neurones qui se trouve dans la bibliothèque de Keras de Tensorflow. ResNet50 est un modèle pré-entraîné de réseaux de neurones et contient ainsi les poids et les biais des *train datasets* de plus d'un million d'images sur lesquels il s'est entraîné. Ce réseau de neurones ResNet50 est constitué de 50 layers et peut classifier des images dans 1000 catégories différentes d'objets comme des claviers, souris, stylos et plusieurs animaux ainsi que les êtres humains. Ce type de réseau de neurones est ainsi 8 fois plus dense que les réseaux de neurones VGG tout en ayant une moindre complexité.

Ceci reste possible grâce à la propriété des réseaux de neurones appelée Transfer Learning. Il s'agit effectivement de transférer l'apprentissage effectué sur des *train datasets*

avec un objectif d'apprentissage différent et de l'appliquer sur notre *dataset* pour obtenir des performances proches de 80% dès les premiers apprentissages du modèle.

D'un point de vue plus technique, les réseaux de neurones ResNet présentent des particularités. Ce type de réseau de neurones a provoqué une véritable révolution en permettant d'augmenter le nombre de layers sans impacter fortement le temps de calcul et en évitant d'obtenir un gradient trop petit rendant l'apprentissage long. Cette possibilité a été permise par la mise en place du concept de "skip connection" c'est-à-dire le saut d'un où plusieurs layers (constituant ainsi un bloc). Cette notion introduit ainsi la possibilité pour le modèle d'utiliser un saut qui constitue une fonction identité dans le modèle. Cette fonction identité assure ainsi que le layer suivant, même en ayant sauté un ou plusieurs layers, sera au moins aussi performant que le layer qu'il vient juste d'exécuter parce que de cette manière, chaque layer a accès aux informations du layer qui précède le saut.

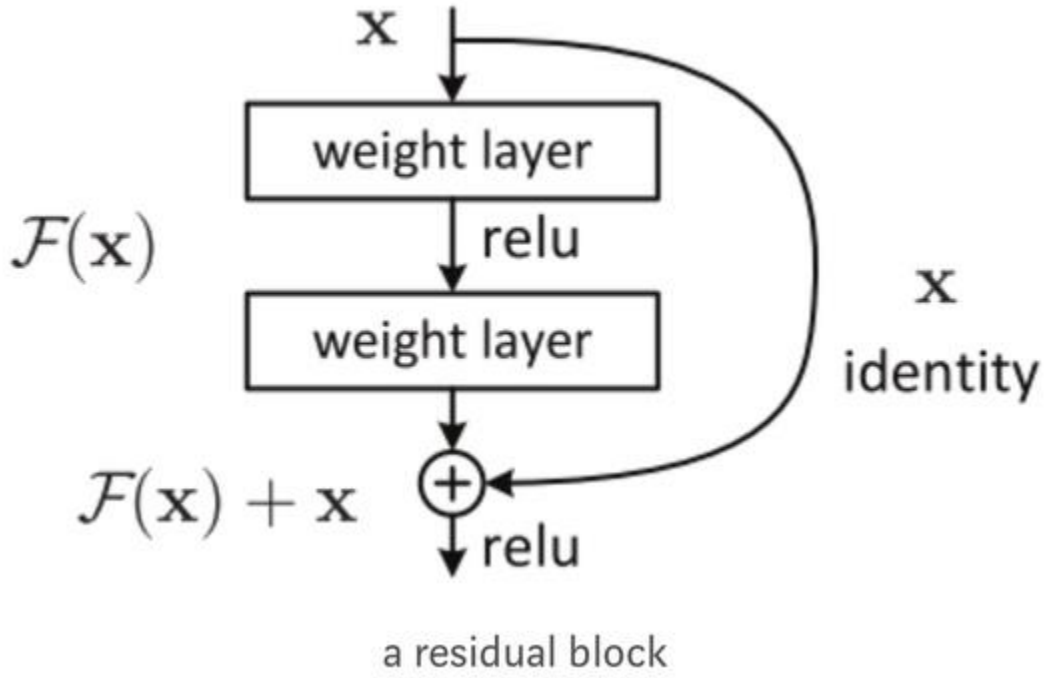


Image d'un bloc résiduel où l'on voit la possibilité dans un réseau de neurones ResNet de sauter des layers.

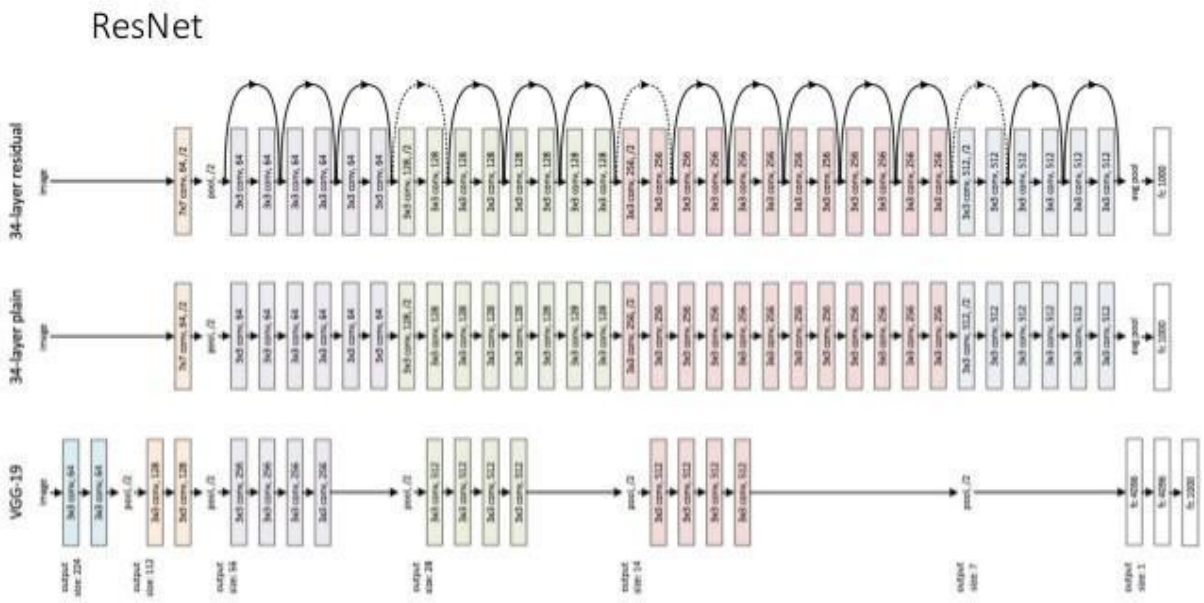


Image du ResNet dans son intégralité

3.3.2. Modèle utilisant ResNet50 dans la pratique

- **Définir le modèle :**

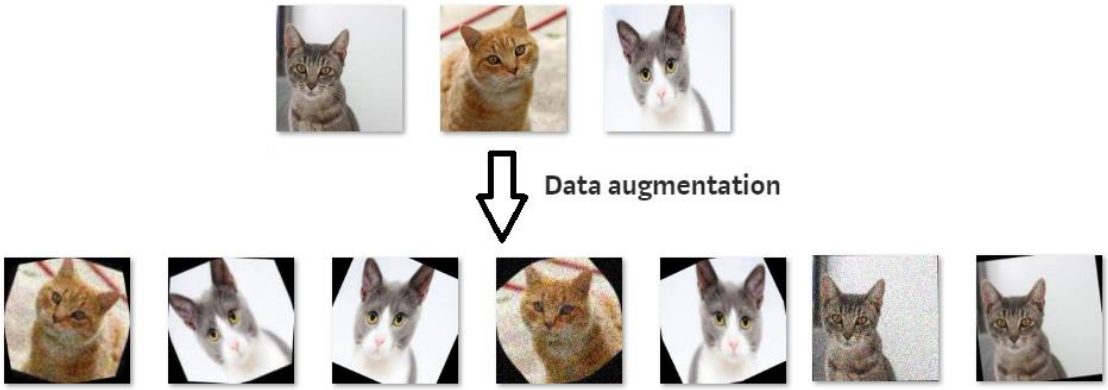
```
from tensorflow.python.keras.applications import ResNet50
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Flatten, GlobalAveragePooling2D

num_classes = 2

my_new_model = Sequential()
my_new_model.add(ResNet50(include_top=False, pooling='avg'))
my_new_model.add(Dense(num_classes, activation='softmax'))

# Say not to train first layer (ResNet) model. It is already trained
my_new_model.layers[0].trainable = False
```

```
Downloading data from https://github.com/keras-team/keras-applications/releases/download/res
net/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
```

- **Compiler le modèle :**

```
my_new_model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
```

- **Ajuster le modèle :** Dans cette étape, on prend les images que l'on a et on les modifie légèrement de manière artificielle. Cela se concrétise par les faire pivoter, changer leur perspective, ou les agrandir. Cette astuce, qui s'appelle l'augmentation des données, nous permet de créer plus de variations sans avoir à télécharger plus de données à l'input mais aussi à apprendre au modèle que le concept qu'on lui apprend n'est pas facteur de ces variations.



Nous avons utilisé Keras comme ci-dessus pour créer un générateur de données d'images. Ensuite, nous avons directement utilisé ce générateur dans notre modèle grâce à la fonction *flow_from_directory* où on peut définir le format des images.

```
from tensorflow.keras.applications.resnet import preprocess_input
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator

image_size = 224
data_generator = ImageDataGenerator(preprocessing_function=preprocess_input)


train_generator = data_generator.flow_from_directory(
        'train',
        target_size=(image_size, image_size),
        batch_size=24,
        class_mode='categorical')

validation_generator = data_generator.flow_from_directory(
        'train',
        target_size=(image_size, image_size),
        class_mode='categorical')

my_new_model.fit_generator(
        train_generator,
        steps_per_epoch=3,
        validation_data=validation_generator,
        validation_steps=1)
```

```
Found 81 images belonging to 2 classes.
Found 81 images belonging to 2 classes.
3/3 [==============================] - 5s 2s/step - loss: 0.8888 - accuracy: 0.5263 - val_lo
ss: 0.5405 - val_accuracy: 0.8125


<tensorflow.python.keras.callbacks.History at 0x7f7ab182bb38>
```

- **Prédiction :** Enfin et surtout, nous pouvons bien sûr utiliser le modèle pour prédire si c'est une fille ou un garçon. Comme nous pouvons le voir dans la sortie, notre modèle a réussi à étiqueter la fille comme une fille puisque la première valeur de

chaque élément de *preds* était inférieur à 0,5 et le garçon comme un garçon, car la valeur était supérieur à 0,5.

```
def read_and_prep_images(img_paths, img_height=image_size, img_width=image_size):
    imgs = [load_img(img_path, target_size=(img_height, img_width)) for img_path in img_path
s]
    img_array = np.array([img_to_array(img) for img in imgs])
    output = preprocess_input(img_array)
    return(output)
```

```
test_data = read_and_prep_images(image_paths)
preds = my_new_model.predict(test_data)
preds[:5]
```

```
array([[0.2349301 , 0.7650699 ],
       [0.6097645 , 0.39023548],
       [0.4592484 , 0.5407516 ],
       [0.04752772, 0.95247227],
       [0.30561057, 0.69438946]], dtype=float32)
```

```
from learntools.deep_learning.decode_predictions import decode_predictions
from IPython.display import Image, display

for i, img_path in enumerate(image_paths[:10]):
    display(Image(img_path, width=200, height=200))
    if preds[i][0] < 0.5:
        print("girl")
    else:
        print("boy")
```

```
girl
```

```
boy
```

3.3.3. Evaluation

Nous avons utilisé cette fonction afin de voir les performances d'un modèle déjà entraîné. Et en effet cela a aidé notre modèle à monter à 81% de précision.

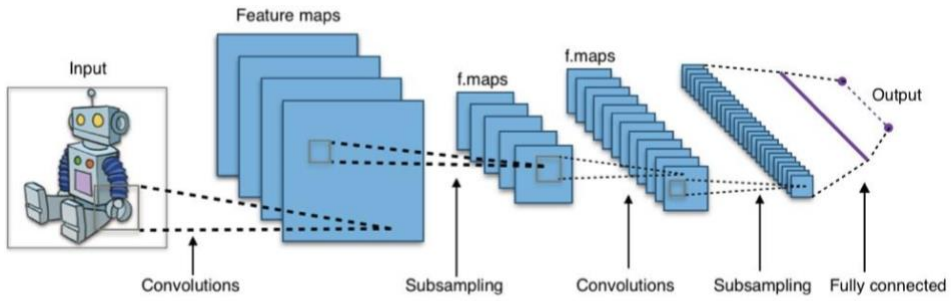
3.4. Modèle utilisant Conv2D

3.4.1. Principe du modèle utilisant Conv2D

Afin de pouvoir comparer les performances obtenues à partir de neurones déjà entraînés, nous avons construit un modèle à partir de rien en utilisant des layers convolutionnels d'un réseau de neurones convolutif. Ce modèle de Deep Learning Conv2D est ainsi l'un des modèles fondamentaux en Deep Learning. Il repose sur le calcul par convolution de matrices. Ce modèle fait partie des réseaux de neurones convolutifs.

Les réseaux de neurones convolutifs sont un type de réseau de neurones acycliques où le motif de connexion entre les neurones est inspiré du cortex visuel des animaux. Il consiste en un empilage multicouche de perceptrons dont le but est de traiter de petites quantités d'informations.

Les neurones convolutifs sont constitués de layers convolutifs qui prennent un input à 3 dimensions, généralement une image avec les 3 couleurs RGB. Les images passent par un filtre qui est en général de dimensions plus petites que l'image. Ce filtre passe en revue une fenêtre de plusieurs pixels en même temps, par exemple 3x3 ou 5x5 pixels et se déplace sur toute l'image jusqu'à avoir scanné tous les pixels de l'image. L'opération de convolution calcule le produit des valeurs des pixels avec les poids définis dans ce filtre. Après ces convolutions, on analyse des sous-ensembles de ces résultats de convolutions selon le schéma suivant :



Convolution Neural Network

Source: Wikipedia

On réduit ainsi petit à petit la taille des ensembles que l'on analyse tout en augmentant la largeur du réseau de neurones. Enfin, en obtenant des ensembles suffisamment petits, on peut les relier (et former une ou des couches entièrement connectées) et obtenir l'output du modèle.

3.4.2. Modèle utilisant Conv2D dans la pratique

- **Définir le modèle :**

  Un réseau de neurones à convolution est essentiellement composé de 4 parties :

  • Convolution
  • Non-linéarité (ReLu)
  • Pooling
  • Classification

  Le type de modèle que nous utiliserons est séquentiel. Nos 3 premières couches sont des couches Conv2D. Ce sont des couches de convolution qui traiteront nos images d'entrée, qui sont considérées comme des matrices bidimensionnelles.

  32 dans les deux premières couches et 64 dans la troisième couche sont le nombre de nœuds dans chaque couche. Ce nombre peut être ajusté pour être supérieur ou inférieur, selon la taille de l'ensemble de données. La taille du noyau est la taille de la matrice de filtre pour notre convolution. Donc, une taille de noyau de 3 signifie que nous aurons une matrice de filtre 3x3.

  L'activation est la fonction d'activation de la couche. La fonction d'activation que nous utiliserons pour nos 2 premières couches est la ReLU, ou Rectified Linear Activation. Cette fonction d'activation s'est avérée bien fonctionner dans les réseaux de neurones.Entre les couches Conv2D et la couche dense, il y a une couche «Flatten». Flatten sert de connexion entre la convolution et les couches denses.

  «Dense» est le type de couche que nous utiliserons pour notre couche de sortie. Dense est un type de couche standard qui est utilisé dans de nombreux cas pour les réseaux de neurones.

  Nous aurons 2 nœuds dans notre couche de sortie. L'activation est «softmax». Softmax fait la somme de la sortie jusqu'à 1 afin que la sortie puisse être interprétée comme des probabilités. Le modèle fera ensuite sa prédiction en fonction de l'option qui a la plus forte probabilité.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(image_size, image_size, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

- **Compiler le modèle :**

Ensuite, nous devons compiler notre modèle. La compilation du modèle prend trois paramètres : optimiseur, perte et métriques.

Nous utiliserons *categorical_crossentropy* pour notre fonction de perte. Il s'agit du choix le plus courant pour la classification. Un score inférieur indique que le modèle fonctionne mieux.

```
# COMPILE
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

- **Ajuster le modèle :**

Une fois le modèle défini,

Les ensembles de données du monde réel sont trop volumineux pour tenir dans la mémoire. Ils nous obligent à effectuer une augmentation des données pour éviter le surajustement et augmenter la capacité de notre modèle à se généraliser.

Dans cette situation, nous devons utiliser encore une fois *fit_generator* et *ImageDataGenerator.*

```
# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

# this is the augmentation configuration we will use for testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1./255)

# this is a generator that will read pictures found in subfolers of 'data/train', and indefini
tely generate
# batches of augmented image data
train_generator = train_datagen.flow_from_directory(
        'train',  # this is the target directory
        target_size=(image_size, image_size),
        batch_size=batch_size,
        class_mode='categorical')  # since we use binary_crossentropy loss, we need binary lab
els
```

```
# this is a similar generator, for validation data
validation_generator = test_datagen.flow_from_directory(
        'val',
        target_size=(image_size, image_size),
        batch_size=batch_size,
        class_mode='categorical')
# TRAINING
model.fit_generator(
        train_generator,
        steps_per_epoch=3,
        epochs=10,
        validation_data=validation_generator,
        validation_steps=16)
```

3.4.3. Evaluation

Après 10 époques, nous avons atteint une précision de 81,82% sur notre ensemble de validation.

```
Epoch 1/10
3/3 [==============================] - 1s 315ms/step - loss: 0.6572 - accuracy: 0.8182
Epoch 2/10
3/3 [==============================] - 1s 417ms/step - loss: 0.6509 - accuracy: 0.7708
Epoch 3/10
3/3 [==============================] - 1s 233ms/step - loss: 0.5369 - accuracy: 0.7879
Epoch 4/10
3/3 [==============================] - 1s 440ms/step - loss: 0.5877 - accuracy: 0.7500
Epoch 5/10
3/3 [==============================] - 1s 241ms/step - loss: 0.4790 - accuracy: 0.7879
Epoch 6/10
3/3 [==============================] - 1s 424ms/step - loss: 0.5709 - accuracy: 0.7708
Epoch 7/10
3/3 [==============================] - 1s 273ms/step - loss: 0.4905 - accuracy: 0.8485
Epoch 8/10
3/3 [==============================] - 1s 388ms/step - loss: 0.5526 - accuracy: 0.7292
Epoch 9/10
3/3 [==============================] - 1s 416ms/step - loss: 0.5718 - accuracy: 0.7500
Epoch 10/10
3/3 [==============================] - 1s 272ms/step - loss: 0.4891 - accuracy: 0.8182
```

3.5. Limites des modèles

Les modèles de *Deep Learning* bien entraînés analysent correctement la majorité des inputs. Cependant, il est très rare voire impossible d'obtenir un taux d'erreur nul et plusieurs biais peuvent être introduits dès l'apprentissage automatique du modèle sur le *train dataset*.

En effet, l'*overfitting* ou surentraînement constitue le principal écueil à éviter lors de l'entraînement du modèle. Par définition, c'est une analyse statistique qui correspond trop étroitement ou exactement à un ensemble particulier de données. Cette analyse peut ainsi ne pas correspondre à de nouvelles données ou ne pas prévoir de manière fiable les observations futures. Ce modèle surajusté est un modèle statistique qui contient plus de paramètres que ne peuvent le justifier les données.

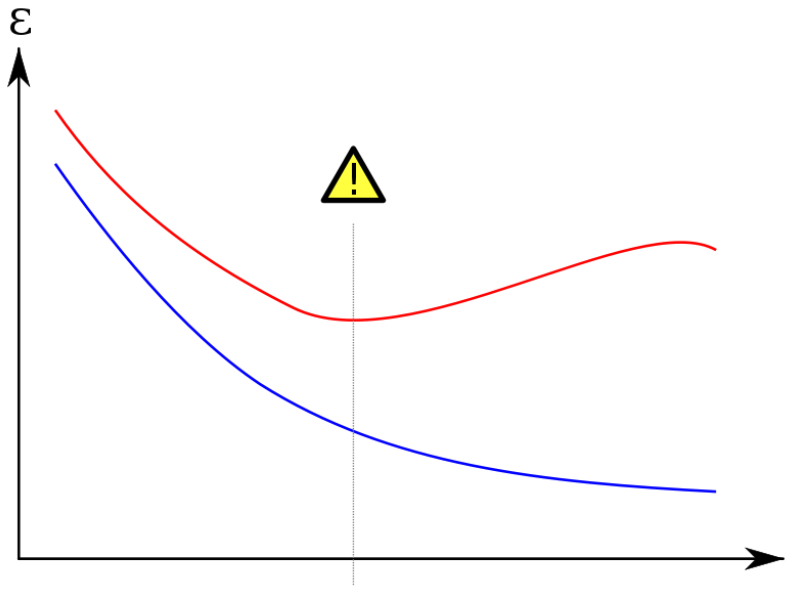


Image caractérisant l'effet d'*overfitting*

Comme illustré ci-dessus, l'*overfitting* se caractérise aussi par une recrudescence du taux d'erreur sur le *validation dataset* à partir d'un certain seuil d'apprentissage alors qu'en bleu, le taux d'erreur sur le *train dataset* continue de diminuer. La capacité d'apprentissage du modèle est alors limitée à une "mémorisation" des *features* du *dataset* sans aucune capacité de généralisation. Cela peut résulter d'un choix pas assez restreint ou contraint du modèle par un mauvais dimensionnement de la structure utilisée pour classifier les éléments, dimensionnement trop grand permettant cette mémorisation (trop de *layers* par exemple dans le modèle).

Un deuxième écueil à éviter se trouve dans la structure de données qui n'est pas adaptée. Nous avons pu être confrontés à des difficultés notamment dans l'utilisation du réseau de neurones Conv2D. Certaines images ont été tronquées alors que d'autres ont vu certains de leurs pixels remplacés par du gris pour pouvoir les analyser. Ce problème vient du mauvais dimensionnement au préalable des images. Afin d'obtenir le même format d'analyse, nous avons été obligés de tronquer voire de remplir artificiellement des images par du gris pour les analyser, introduisant ainsi naturellement des biais dans les données et par conséquent dans l'apprentissage du modèle.

Pour y remédier, nous avons dû identifier les images les plus proches du format que nous voulions utiliser et supprimer les autres qui posaient problème. Face à une diminution du nombre de données d'entrée, nous avons eu recours à la data augmentation en dupliquant plusieurs fois une même image pour entraîner le modèle et grossir le *train dataset*.

IV- Estimation des coûts (en heures)

Pour clôturer ce rapport, nous allons estimer les coûts en temps de travail du développement de l'ensemble du code qui a été produit.

Pour réaliser l'analyse exploratoire des données, il nous a fallu environ 40 heures, en prenant aussi une remise à niveau en Python (environ 6-8 heures) et l'apprentissage des techniques de *machine learning* (6-8 heures) et *deep learning* (6-8 heures).

Pour réaliser les modèles de *machine learning*, il nous a fallu environ 24 heures.

Enfin pour réaliser les modèles de *deep learning*, il nous aura fallu 32 heures.

Ainsi, nous pouvons estimer le total d'heures d'apprentissage et de codage d'environ 96 heures.

V- Conclusion

En conclusion de ce projet d'application, il me faut insister sur plusieurs points majeurs qui ont été mis en exergue et qui sont fondamentaux pour mener une bonne étude de données et élaborer correctement un modèle d'apprentissage automatique :

- Connaître ses données :

La connaissance des données est primordiale dans l'apprentissage supervisé afin d'éviter tous les biais statistiques qui compromettent le bon entraînement du modèle.

- Choisir correctement son modèle d'apprentissage automatique :

Chaque modèle d'apprentissage automatique a ses forces et ses faiblesses selon les données à analyser. Choisir le meilleur modèle d'apprentissage automatique découle ainsi naturellement d'une bonne connaissance de ses données ainsi que des modèles à disposition. Connaître les modèles impliquent également connaître le paramétrage requis afin d'obtenir les meilleures performances.

- Connaître les limites de son modèle et savoir comment les résoudre :

Chaque modèle a ses faiblesses et ses limites. Plusieurs méthodes permettent d'y remédier. Nous pouvons par exemple combiner différents modèles d'apprentissage automatique spécialisés dans une tâche spécifique permettant ainsi d'augmenter de quelques points la performance du modèle.

Synthèses personnelles :

Canran CHEN :

Dans la première partie, nous nous sommes renseignés sur les différentes étapes de la construction de modèles comme la construction de modèles, l'évaluation et enfin la prédiction. Nous avons également appris les différentes étapes de la création de modèles. Ce jeu de données est une très bonne entrée en matière. Il nous a permis de nous former rapidement au langage Python et à ses bibliothèques dédiées au calcul ainsi qu'au *machine learning*. De plus, les problématiques d'analyse prédictives sont abordées sur un exemple aux difficultés limitées : nombre de variables assez faibles, etc.

Au travers du deuxième sujet, nous avons été exposés aux problématiques concrètes qui se posent régulièrement en *machine learning*, telles que la taille exubérante des données, la puissance limitée de nos outils de travail. Ce fut difficilement gérable au commencement du projet mais nous avons ensuite réussi à contourner habilement ces problèmes, notamment en exerçant notre algorithme sur un sous-ensemble des données et non sur son intégralité. Évidemment, cela réduisait l'information disponible et diminuait donc la qualité possible de nos algorithmes, cependant cela nous a permis un premier travail de base sur les données et la manière de les utiliser.

Personnellement, je ressors grandi de ce projet d'application. Ce projet long de 4 mois est une bonne expérience, il m'a apporté beaucoup, tant au niveau technique qu'en terme de gestion de projet même si nous avons été seulement deux étudiants. Nous avons confirmé le fait que la communication est primordiale lorsque l'on travaille ensemble, soit un dialogue par messagerie instantanée ou une entrevue en face à face. Le sujet intéressant m'a donné envie de travailler. La bonne communication a facilité ma compréhension. Que du positif, une expérience à renouveler !

Clément GUILLAUME :

Ce projet d'application a constitué pour ma part une première approche riche et diversifiée de la data science et des techniques de *machine learning*. Beaucoup de thématiques ont été abordées offrant ainsi un bel ensemble assez synthétique à mon avis de ces domaines. Ce projet m'encourage a développé mes compétences en Python, un des langages de prédilection pour le développement de modèles de *machine learning* ainsi que l'approfondissement des différents types de réseaux de neurones et autres modèles possibles de *deep learning* et plus largement de *machine learning*.

Dans ce projet, nous avons été deux étudiants. Nous avons travaillé ensemble jusqu'au bout et je pense que ce projet m'a donné une nouvelle expérience de collaboration, cette fois-ci en équipe très réduite et avec une certaine dimension multiculturelle. Je pense que ce projet et son environnement de travail nous ont donné un premier aperçu d'un travail en équipe sur un projet informatique et je pense que ce type de projet d'application ne peut qu'être bénéfique à tout étudiant désirant d'en apprendre davantage dans les domaines de la data science et du *machine learning*.