



A.G.M

EILCO
ÉCOLE D'INGÉNIEURS
DU LITTORAL CÔTE D'OPALE

COMPTE RENDU

30/03/2024

L'ESTIMATION DE L'ÂGE DES OTOLITHES PAR DES APPROCHES D'APPRENTISSAGE AUTOMATIQUE

Groupe:

Achiba Ahmed
Bougra Mohamed
EL Ouartl Mouad
El Okri Amine
Zahdi Ghita

encadré par:

Emilie Poisson-Caillault

2023/2024

Prétraitement des données

Après avoir divisé la base de données en deux parties (right.csv ,right_left.csv) : les otolithes droits et les otolithes gauches, nous avons commencé à entraîner les algorithmes de segmentation d'images en utilisant l'apprentissage approfondi pour estimer l'âge des poissons, en nous basant dans un premier temps sur les otolithes droits. Au cours de notre travail, nous avons rencontré plusieurs défis, notamment dans la base de données d'images, parmi eux :

Problème de l'extension des images

Les images données ont pour extension .tif, cependant, la bibliothèque Keras utilisée pour entraîner les algorithmes de deep learning ne prend en charge que les extensions .jpeg, .jpg, .png, .bmp et .gif. Il existe une seule bibliothèque qui permet de lire des images avec l'extension .tif, mais elle ne permet pas d'entraîner nos modèles.

Par conséquent, nous avons décidé de convertir les images au format .png afin de ne pas perdre les détails des images et de réduire la taille de la base de données.

```
datas=data['Reference_PC']
dossier='./right/'
out_folder='./right_png/'
for j in lost:
    e=datas[j]
    file=dossier+e+'.tif'
    out_file=out_folder+e+'.png'
    index=e
    img = tiff.imread(file).transpose(1,2,0)
    imageio.imwrite(out_file, img)
    #resized_img=cv2.resize(img, (256, 256), interpolation=cv2.INTER_AREA)
    imgs.append(img)
    img=None
```

Ce processus nous a pris beaucoup de temps.

454m 50.6s

⊗ 120m 23.2s

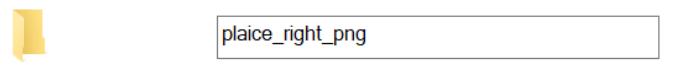
La base de données devient moins volumineuse.

Avant



Type : Dossier de fichiers
Emplacement : C:\Users\pc\OneDrive\Desktop\EILCO\ProjetTech\right
Taille : 11,9 Go (12 810 582 419 octets)

Après



Type : Dossier de fichiers
Emplacement : C:\Users\pc\OneDrive\Desktop\EILCO\ProjetTech\plaice_right_png
Taille : 5,88 Go (6 317 926 965 octets)

Transférer chaque image dans son sous-répertoire correspondant

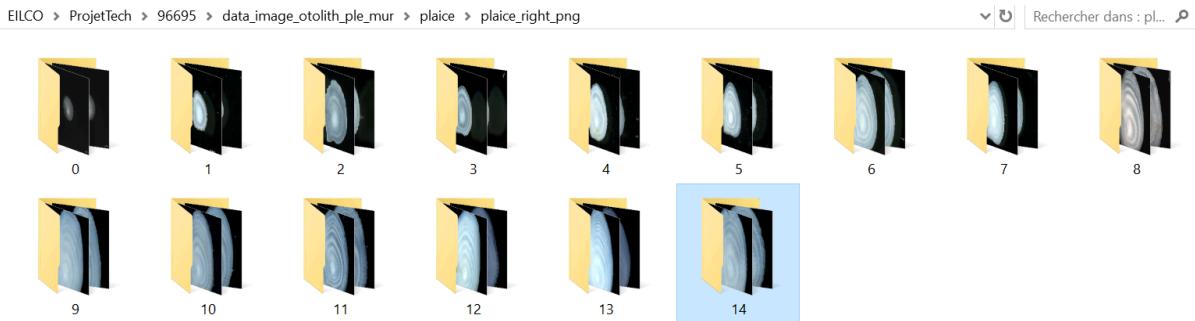
En effet Keras parcourt récursivement le répertoire de la base de données et collecte les images ainsi que leurs étiquettes en fonction des noms de sous-répertoires.

En d'autres termes, chaque image trouvée dans un sous-répertoire est automatiquement associée à l'étiquette correspondant au nom de ce sous-répertoire. Par conséquent, l'utilisation de fichier '**right.csv**' pour étiqueter les images n'est plus nécessaire.

```
#####ORGANIZE DATABASE#####
import shutil
import os

def move_file(src_fold, dest_fold, filename):
    source_path = os.path.join(src_fold, filename)
    destination_path = os.path.join(dest_fold, filename)
    try:
        shutil.move(source_path, destination_path)
    except FileNotFoundError:
        print(f"fichier '{filename}' n'est pas trouve dans '{src_fold}' .")
    except PermissionError:
        print(f"probleme de permission '{filename}' .")

src_fold = './right_png'
dest_fold = './plaice_right_png/'
```



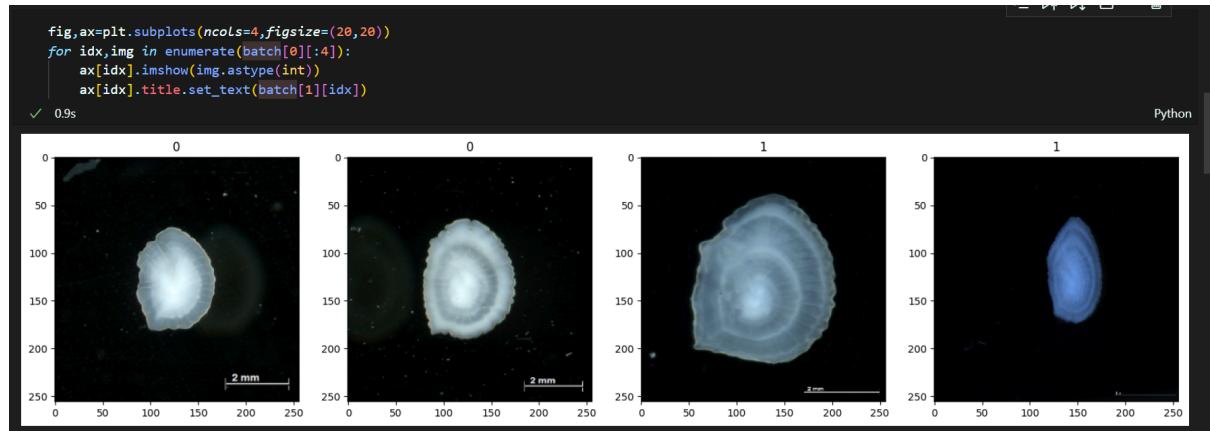
Redimensionnement des images

a Même dimension

Les images doivent avoir la même dimension lorsqu'elles sont utilisées pour l'entraînement de modèles de réseaux de neurones, en particulier avec des architectures conventionnelles, pour plusieurs raisons :

- **Entrée cohérente pour le modèle** : Les modèles de réseaux de neurones, en particulier les réseaux convolutionnels (CNN), sont conçus pour traiter des données avec des dimensions spécifiques en entrée. Des dimensions différentes peuvent entraîner des problèmes lors du passage des images à travers les différentes couches du réseau.
- **Matrices d'entrée homogènes** : Les images sont généralement représentées sous forme de matrices, où chaque pixel correspond à une valeur dans la matrice. Pour que le modèle puisse traiter efficacement ces matrices, elles doivent toutes avoir la même taille.
- **Réduction de la complexité** : Lorsque les images ont des dimensions différentes, cela complique la gestion des couches du réseau, notamment en termes de nombre de paramètres à entraîner. Des dimensions cohérentes simplifient le processus d'entraînement et aident à réduire la complexité du modèle.
- **Compatibilité avec les opérations de traitement d'images** : De nombreuses opérations de traitement d'images, telles que la convolution et le pooling, nécessitent des entrées de taille fixe pour fonctionner correctement. Des dimensions d'images variables peuvent rendre ces opérations plus complexes à mettre en œuvre.

Pour notre cas nous allons opter la taille standard(256*256) puisque les images de grande taille nécessitent également plus de mémoire et de puissance de calcul, ce qui peut augmenter les exigences en termes de ressources.



```

batch[0].shape
✓ 0.0s
(32, 256, 256, 3)

```

les images sont maintenant de dimension 256*256 sur 3 channels 'rgb'

a Taille des images doivent être inférieure à 512 ko

Un des problèmes rencontrés lors de l'entraînement des données avant redimensionnement des images est que Keras ne supporte que des images de dimensions inférieures à 512 ko, d'où l'importance du redimensionnement des images.

```

Detected at node decode_image/DecodeImage defined at (most
<stack traces unavailable>
Invalid PNG data, size 524417
[[{{node decode_image/DecodeImage}}]]
[[IteratorGetNext]] [Op:_inference_one_step_on_i

```

Calculatrice
≡ Scientifique

$$524417 \div 1024 =$$

512,1259765625

```

batch[0][0].size
[11] ✓ 0.0s
... 196608

```

La mise à l'échelle des données (Data Scaling)

Lors de l'entraînement de modèles de segmentation d'images, il est essentiel d'avoir des données d'entrée bien normalisées pour assurer la stabilité de l'apprentissage. Des données mal mises à l'échelle peuvent entraîner des instabilités numériques et rendre l'entraînement du modèle plus difficile.

Dans le cas de la segmentation d'images, chaque pixel de l'image est une caractéristique distincte. En normalisant les valeurs des pixels, on s'assure que chaque pixel contribue de manière équitable au processus d'apprentissage, ce qui peut conduire à une meilleure qualité de segmentation.

```
scaled_data_iterator=scaled_data.as_numpy_iterator()
scaled_batch=scaled_data_iterator.next()

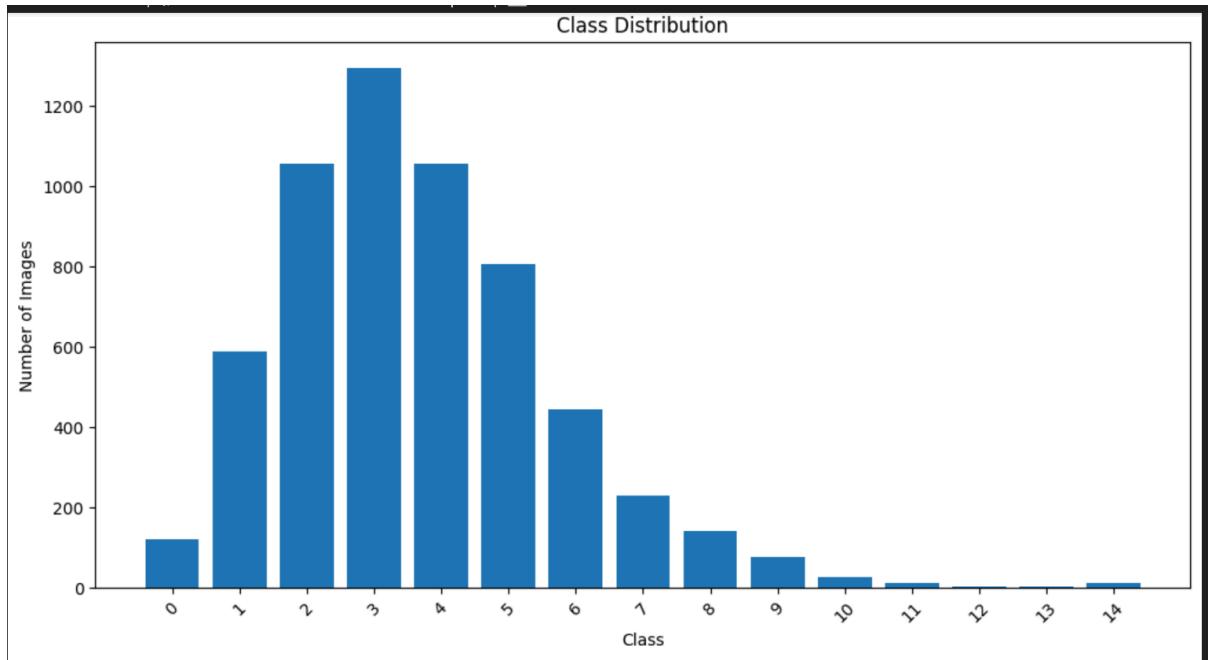
[26] ✓ 0.0s

scaled_batch[0].max()
[27] ✓ 0.0s
...
1.0

scaled_batch[0].min()
[28] ✓ 0.0s
...
0.0
```

L'équilibrage de la base de données

En effet, la base de données sur laquelle nous travaillons est déséquilibrée ; on observe une grande disparité dans la distribution des images entre les différentes classes, ce qui peut entraîner un surajustement. Par conséquent, nous devons utiliser des techniques d'équilibrage des bases de données pour rendre les classes égales.



Pour notre cas, nous avons utilisé comme technique d'équilibrage , sur-échantillonnage (over-sampling), cette technique consiste à augmenter le nombre d'échantillons de la classe minoritaire en ajoutant des duplcatas des échantillons existants ou en générant de nouveaux échantillons synthétiques.

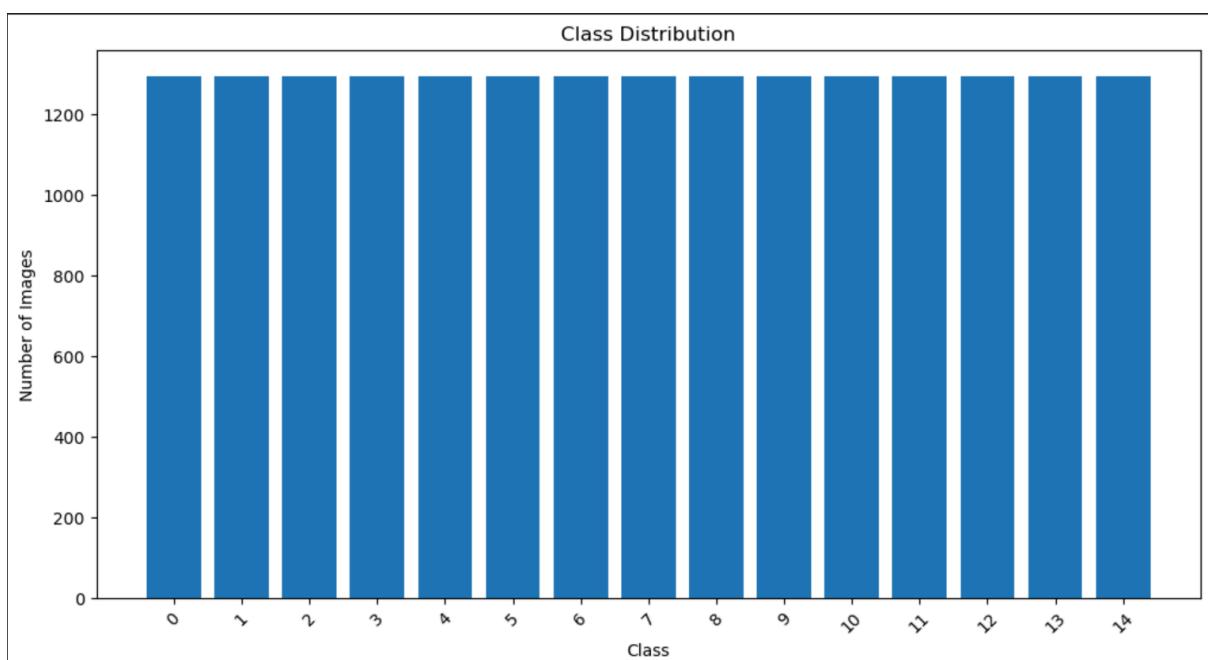


Image non étiquetées

La classe représentant l'ensemble des images non étiquetées est composée de 14 fichiers.

Après l'entraînement de notre modèle, nous allons appliquer ce modèle sur ces images pour prédire leurs classes, puis nous réentraînerons à nouveau le modèle avec ces nouvelles données.

Données d'entraînement, de validation et de test

Les données d'entraînement, de validation et de test sont les trois ensembles principaux utilisés dans le processus de développement et d'évaluation des modèles d'apprentissage automatique.

- **Données d'entraînement** : Ces données sont utilisées pour entraîner le modèle, c'est-à-dire pour ajuster les poids des paramètres du modèle afin qu'il puisse apprendre à partir des données et faire des prédictions.

Pour notre cas, la taille des données d'entraînement est de 70% de l'ensemble des données .

- **Données de validation** : Ces données sont utilisées pour évaluer les performances du modèle pendant l'entraînement et pour prendre des décisions sur les hyperparamètres du modèle, tels que le taux d'apprentissage ou la taille du lot. Les données de validation sont utilisées pour surveiller la capacité du modèle à généraliser aux données inconnues tout en évitant le surajustement.

Pour notre cas, la taille des données d'entraînement est de 20% de l'ensemble des données

- **Données de test** : Une fois que le modèle a été entraîné et validé sur les données d'entraînement et de validation, il est évalué sur les données de test pour obtenir une estimation impartiale de sa performance finale.

Pour notre cas, la taille des données d'entraînement est de 10% de l'ensemble des données .

```
train_size=int(len(n_data)*.7)
val_size=int(len(n_data)*.2)+1
test_size=int(len(n_data)*.1)

✓ 0.0s

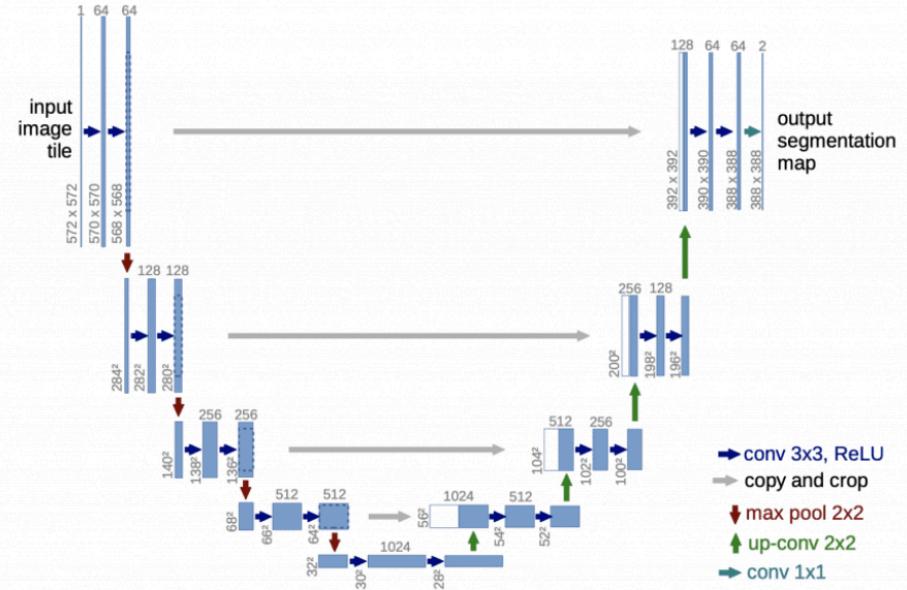
train_size+val_size+test_size
✓ 0.0s
52

n_train=n_data.take(train_size)
n_val=n_data.skip(train_size).take(val_size)
n_test=n_data.skip(train_size+val_size).take(test_size)
✓ 0.0s
```

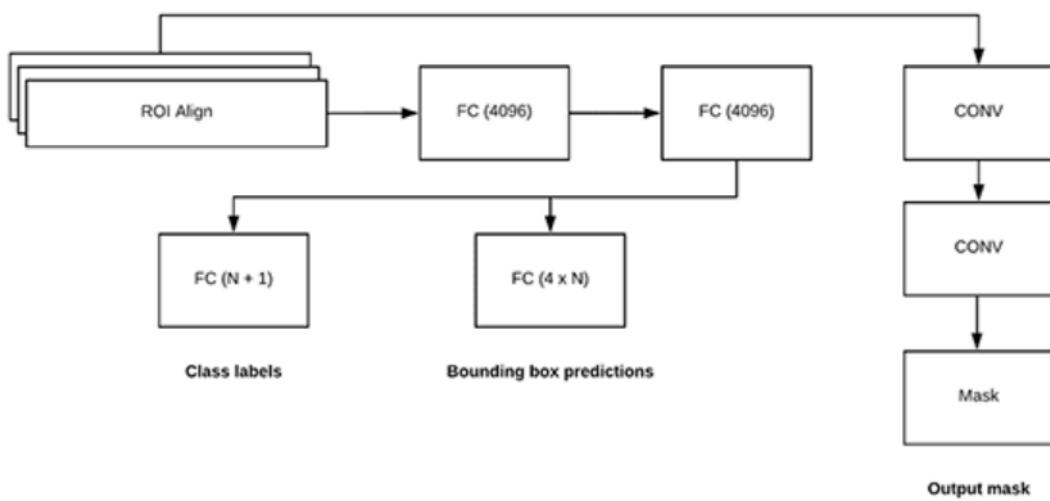
Modèles d'apprentissage profond pour segmenter les images

Les algorithmes d'apprentissage approfondi utilisés pour segmenter les otolithes afin d'estimer l'âge des poissons sont :

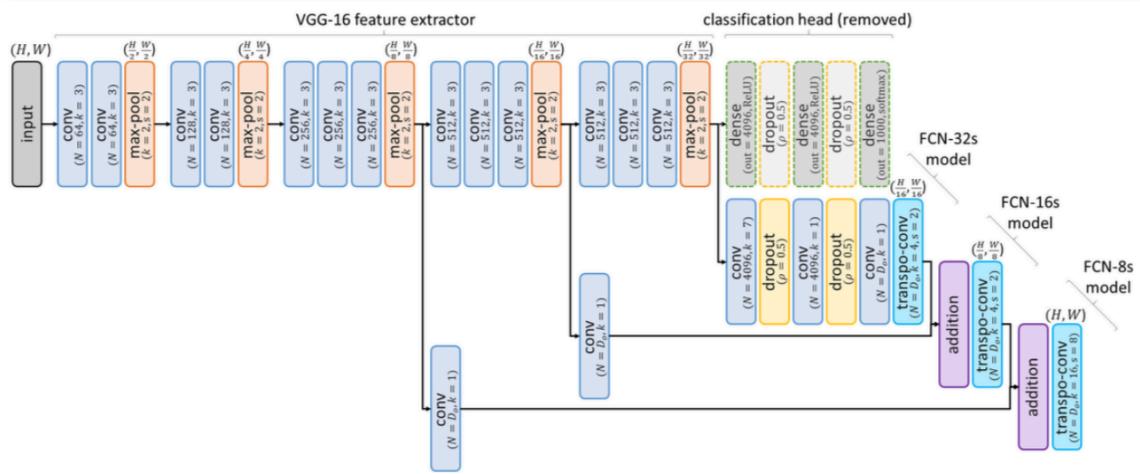
- **U-Net** : Il utilise une architecture en forme de U qui combine des couches de contraction (encodeurs) et d'expansion (décodeurs) pour capturer à la fois les caractéristiques globales et locales de l'image.



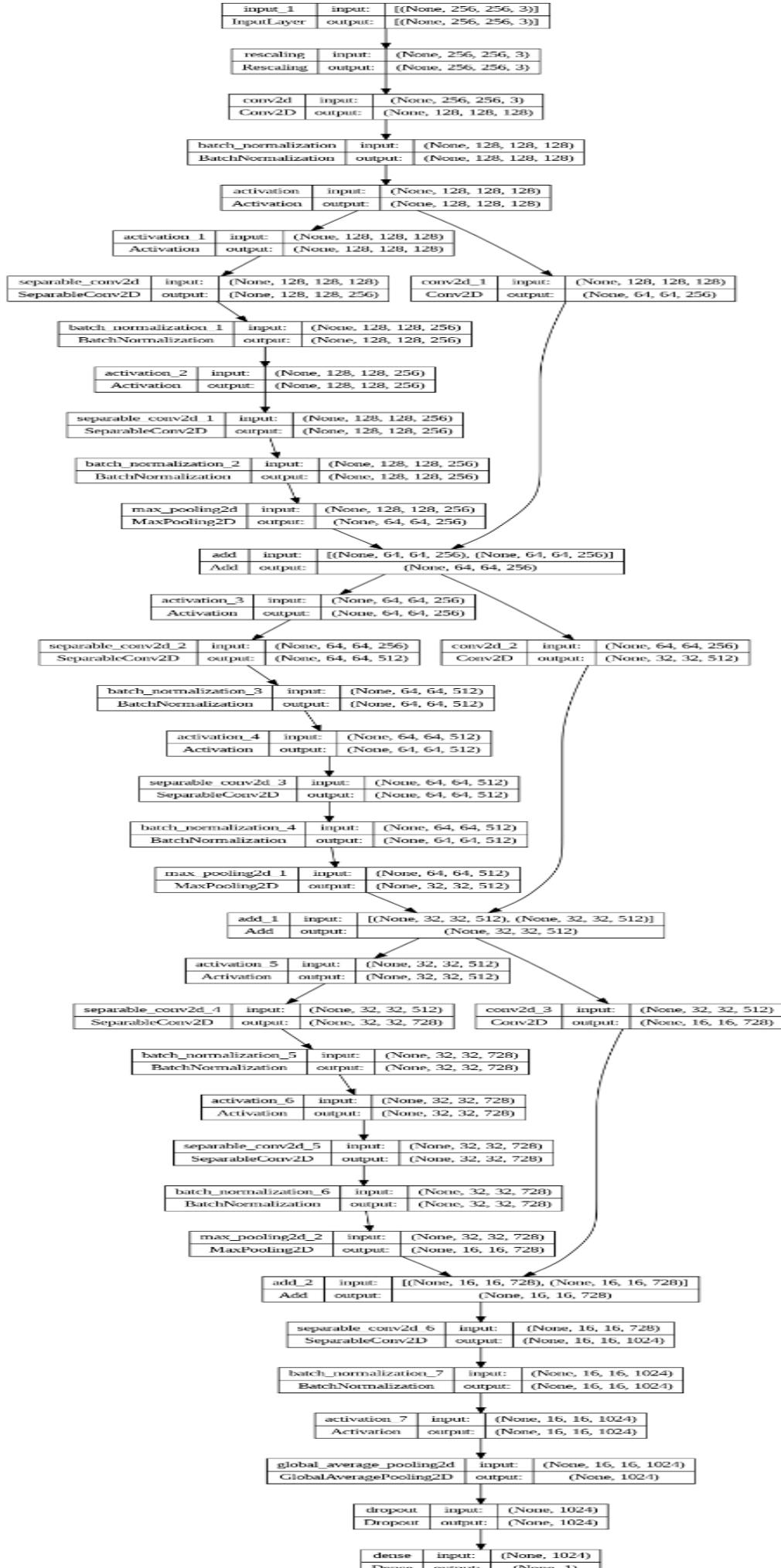
- **Mask R-CNN** : Basé sur l'architecture R-CNN, Mask R-CNN ajoute une branche de segmentation aux régions d'intérêt détectées par le réseau. Cela permet de générer des masques de segmentation précis pour chaque instance d'objet dans une image.



- **FCN (Fully Convolutional Network)** : FCN prédit une carte de segmentation pixel par pixel. Il utilise des couches de convolution transposables (ou des upsampling layers) pour agrandir progressivement la carte de segmentation.

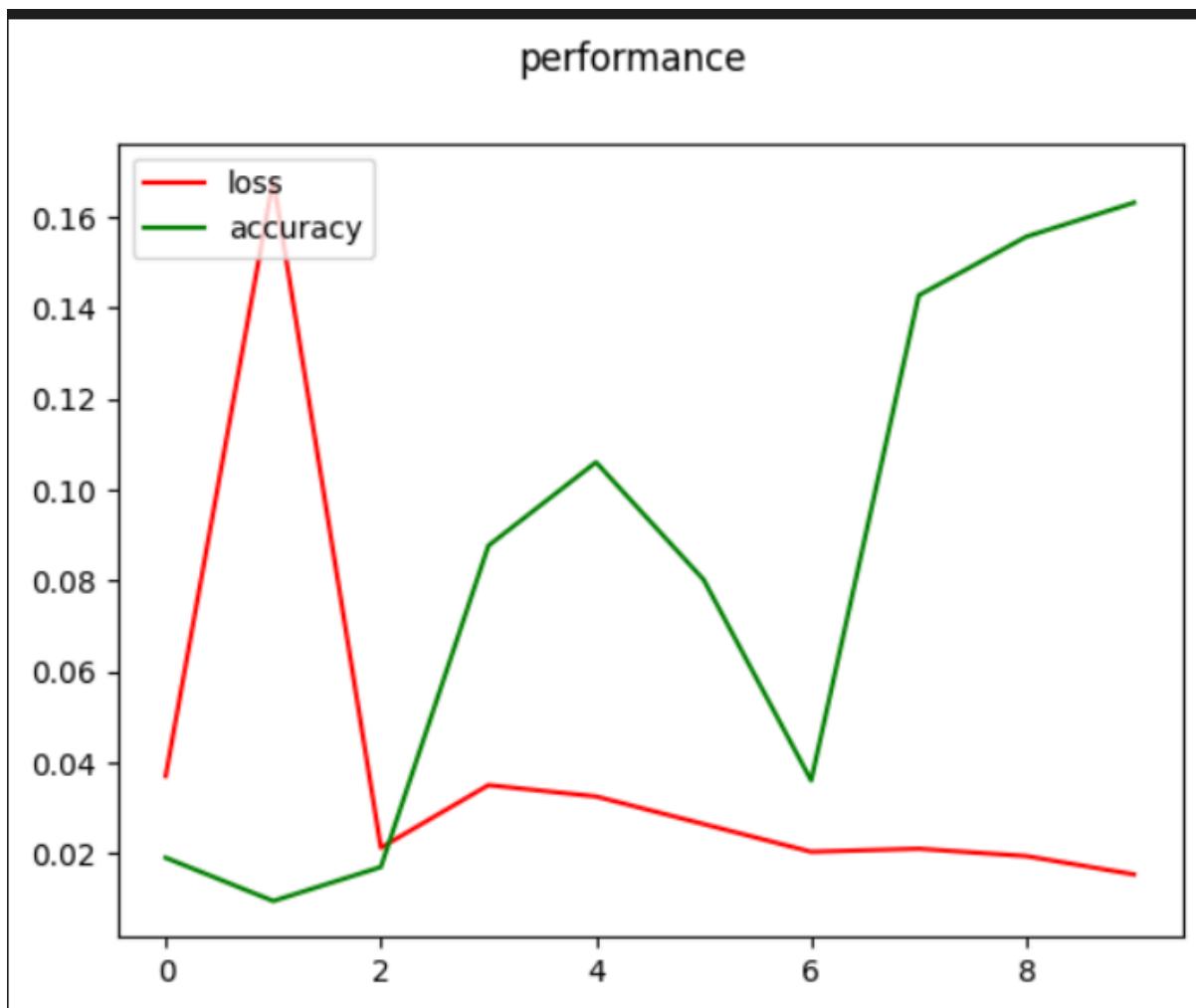


Dans un premier temps, nous avons implémenté l'algorithme U-Net, voilà l'architecture du modèle:



Performance du modèle

Nous avons entraîné notre modèle sur un certain nombre d'epochs, où un epoch représente une seule passe complète de l'ensemble des données d'entraînement à travers le modèle. Après chaque epoch (ou après plusieurs epochs), le modèle peut être évalué sur un ensemble de données de validation pour observer son comportement sur des données qu'il n'a pas rencontrées pendant l'entraînement. En réalité, notre modèle est très complexe et la base de données est très volumineuse, ce qui signifie que le modèle nécessite beaucoup de temps pour l'entraînement. Pour cette raison, nous avons initialement choisi un nombre d'epochs relativement petit (10 epochs) afin d'observer le comportement du modèle.



Nous observons qu'après chaque epoch, le modèle s'évalue de manière positive, avec une diminution de la quantité d'erreur entre les prédictions du modèle et les vraies valeurs sur un ensemble de données de validation (val_loss), ainsi qu'une augmentation de la validation accuracy, représentant le pourcentage de prédictions correctes parmi toutes les prédictions effectuées sur cet ensemble de données. Donc, si l'on parvient à trouver le nombre d'epochs optimal, nous aurons le meilleur modèle.

Evaluation du modèle sur l'ensemble des données de test

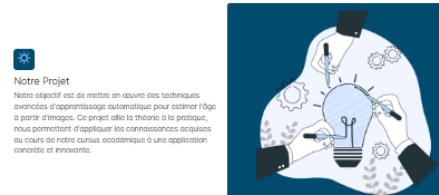
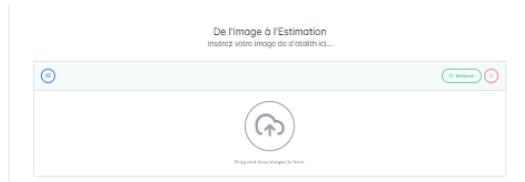
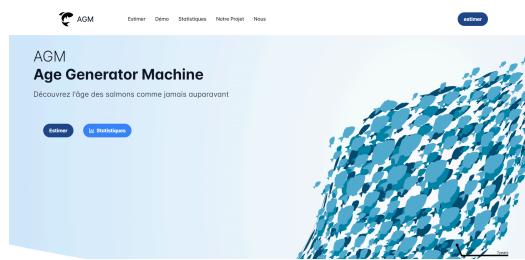
```
▶ [53] print(f'Accuracy: {acc.result().numpy()*100} %')
✓ 0.0s
...
Accuracy: 22.171945869922638 %
```

On observe que la précision sur l'ensemble de données est très faible. Par conséquent, nous devons revoir l'architecture du modèle ou bien tester les données sur un autre modèle d'apprentissage approfondi, tel que Mask R-CNN ou FCN (Fully Convolutional Network).

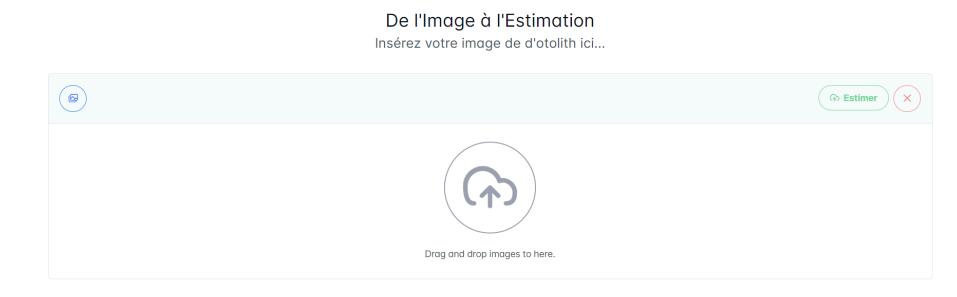
Les Interfaces Web

Dans le cadre de notre projet, la conception et le développement des interfaces web jouent un rôle crucial pour offrir une expérience utilisateur fluide et efficace. Nous avons choisi d'utiliser **Vue.js** comme framework principal pour la construction de nos interfaces web en raison de sa flexibilité, de sa performance et de sa communauté active. En outre, pour les composants de visualisation de données, nous avons opté pour la bibliothèque **PrimeNG** en raison de sa richesse en fonctionnalités et de son intégration harmonieuse avec Vue.js.

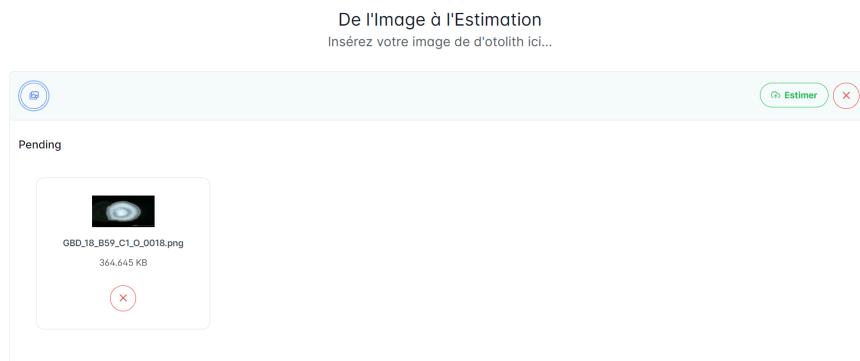
Landing Page:



Section d'estimation de l'âge:



sélectionner votre otolith:



résultat d'estimation :

AGM

Estimer Démo Statistiques Notre Projet Nous

estimer

Estimation d'Age est:

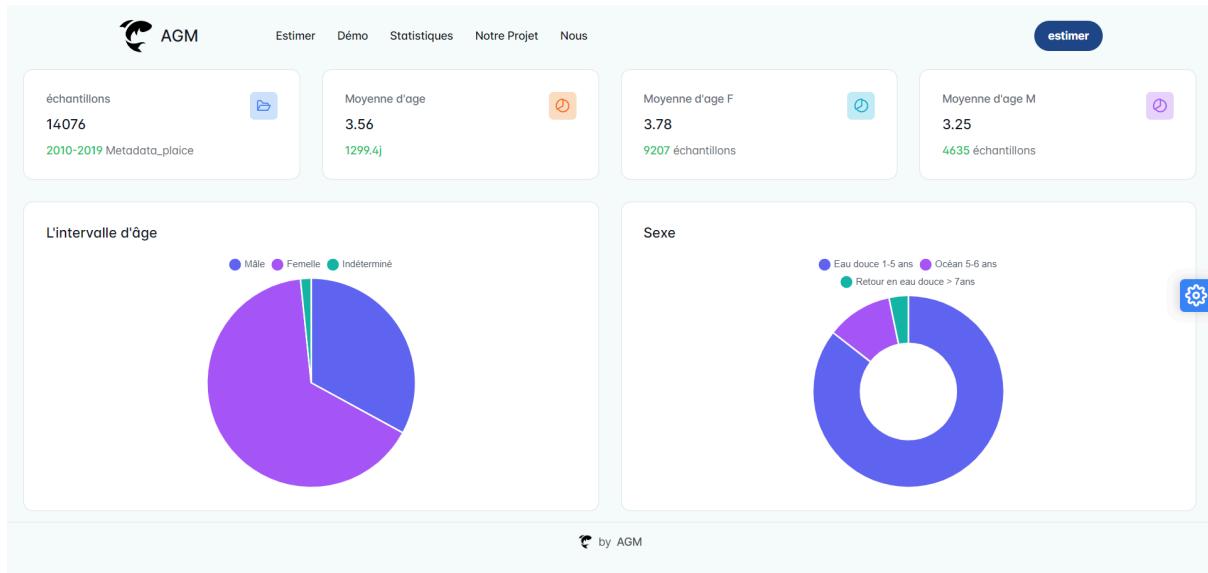
567

jours

0 ⚪ ⚪ ⚪ ⚪ ⚪

by AGM

Page des statistiques:



Pour analyser ces statistiques en utilisant **Python**.

Avant d'effectuer des analyses statistiques, il est essentiel de garantir la qualité et la fiabilité des données. Dans cette optique, un processus de nettoyage des données a été entrepris pour identifier et corriger les éventuelles anomalies ou incohérences dans les jeux de données.

```
# Importer les librairies
# Utiliser la cellule (Ctrl+Enter)
# Je n'ai pas exécuté au cours de cette session
# Les cellules à extraire
# "Age"
# "Sexe"
# "Eau_douce"
# "Ocean"
# "Retour_en_eau_douce"

# Exécuté par mohamed bougra
# le 24 mars 2024 (il y a 7 jours)
# à 0.298 s

# Charger les données
# Utiliser la fonction pd.read_csv()
# Nom du fichier : data_plaice_2010-2019.csv
# Utiliser les colonnes à extraire, séparées par ","
# Utiliser False pour l'index_col
# Utiliser "Age" comme nom pour la colonne

data2 = data2.dropna(subset=["Sexe"])
data2 = data2.loc[(data2["Sexe"] != "-1") & (data2["Age"] != -1.0)]
data2["Sexe"] = data2["Sexe"].replace(["i", "I"], "I")

# Calculer les informations
moyenne_age_total = data2["Age"].mean().round(2)
moyenne_age_sexe = data2.groupby("Sexe")["Age"].mean().round(2)
nombres_par_sexe = (data2["Sexe"].value_counts()).to_dict()
intervalle_1_5 = (data2[(data2["Age"] >= 1) & (data2["Age"] <= 5)].shape[0])
intervalle_5_7 = (data2[(data2["Age"] > 5) & (data2["Age"] <= 7)].shape[0])
intervalle_plus_7 = (data2[data2["Age"] > 7].shape[0])
nombre_lignes = data2.shape[0]

# Créer un dictionnaire contenant les informations
fisher_info = {
    "moyenne_age_total": moyenne_age_total,
    "moyenne_age_sexe": moyenne_age_sexe.to_dict(),
    "nombres_par_sexe": nombres_par_sexe,
    "pourcentage_intervalle_age": {
        "eau_douce": intervalle_1_5,
        "ocean": intervalle_5_7,
        "retour_en_eau_douce": intervalle_plus_7
    },
    "nombre_lignes": nombre_lignes
}

# Écrire les informations dans un fichier JSON
with open("fisher_info.json", "w") as json_file:
    json.dump(fisher_info, json_file, indent=4)
print("Les informations ont été stockées dans fisher_info.json.")
```

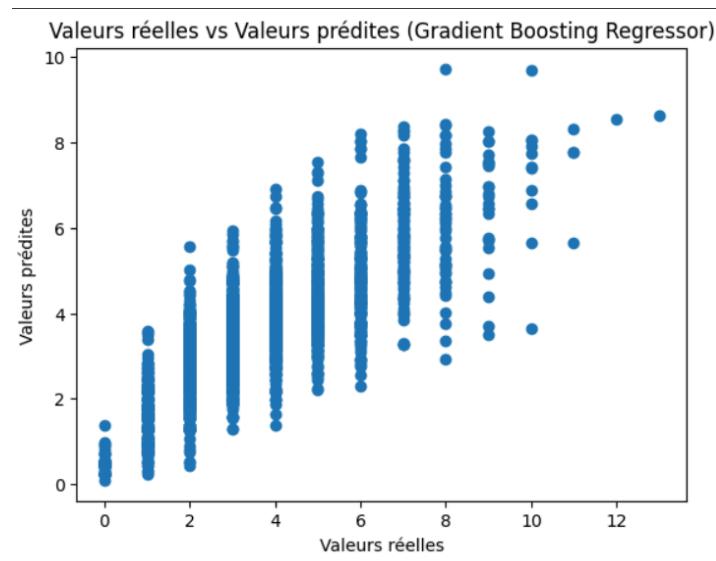
```
1  {
2    "info": {
3      "moyenne_age_total": 3.56,
4      "moyenne_age_sexe": {
5        "F": 3.78,
6        "I": 0.78,
7        "M": 3.25
8      },
9      "nombres_par_sexe": {
10        "F": 9207,
11        "M": 4635,
12        "I": 234
13      },
14      "pourcentage_intervalle_age": [
15        "eau_douce": 11824,
16        "ocean": 1549,
17        "retour_en_eau_douce": 449
18      ],
19      "nombre_lignes": 14076
20    }
21 }
```

Étude de la Relation entre la Taille, le Poids et l'Âge des Poissons

Dans cette partie supplémentaire du projet, nous avons exploré la relation entre la taille, le poids et l'âge des poissons. Pour ce faire, nous avons nettoyé les données en éliminant les valeurs aberrantes et les données manquantes. Ensuite, nous avons utilisé un modèle de régression par gradient boosting pour prédire l'âge des poissons en fonction de leur taille et de leur poids.

Un modèle de **régression par gradient boosting** a été créé et entraîné sur les données d'entraînement. Après avoir entraîné et évalué notre modèle, nous avons constaté une erreur quadratique moyenne (RMSE) de **1.15**. En utilisant ce modèle, nous avons également prédit l'âge d'un nouveau poisson avec une taille de 388 et un poids de 31, obtenant une estimation de **3.0** années.

Cette étude offre un aperçu précieux de la dynamique entre la taille, le poids et l'âge des poissons, ce qui pourrait avoir des implications importantes pour la gestion des populations de poissons et la conservation des ressources marines.



code de nettoyage:

```
import pandas as pd
import numpy as np

# Définir les noms des colonnes à extraire
colonnes_a_extraire = ["Taille", "Poids", "Age"]
# Importer les données
data = pd.read_csv("./metadata_plaice_2010-2019.csv", usecols=colonnes_a_extraire, delimiter=";", index_col=False)
# Convertir la colonne "poids" en type chaîne de caractères
nombre_lignes = data.shape[0]

print("Nombre de lignes dans votre ensemble de données :", nombre_lignes)
data = data.dropna(subset=["Age"])
data = data.dropna(subset=["Poids"])

data["Poids"] = data["Poids"].astype(str)
data["Taille"] = data["Taille"].astype(str)

# Remplacer la virgule par un point
data["Poids"] = data["Poids"].str.replace(",", ".")
data["Taille"] = data["Taille"].str.replace(",", ".") 

data = data.loc[(data["Poids"] != "-1") & (data["Taille"] != " n.") & (data["Age"] != -1.0)]
# Handle potential errors during conversion (optional)
def convert_to_numeric(x):
    try:
        return pd.to_numeric(x)
    except ValueError:
        # Handle non-numeric values (e.g., remove row, impute, or log appropriate message)
        return np.nan # Replace with your preferred handling for non-numeric values

data["Taille"] = data["Taille"].apply(convert_to_numeric)
data["Poids"] = data["Poids"].apply(convert_to_numeric)
data = data.dropna(subset=["Poids"])
# Check for successful conversion and missing values
print(data.dtypes) # Verify data types
print(data.isnull().sum()) # Check for missing values
# Enregistrer les données nettoyées
data.to_csv("data_nettoyé.csv")
```

code de prédition:

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error

# Sélectionner les caractéristiques (taille et poids) et la cible (âge)
X = data[['Taille', 'Poids']]
y = data['Age']

# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Créer et entraîner le modèle Gradient Boosting Regressor
model = GradientBoostingRegressor(n_estimators=500, learning_rate=0.1, max_depth=3, random_state=42)
model.fit(X_train, y_train)

# Prédire l'âge des poissons dans l'ensemble de test
predictions = model.predict(X_test)

# Calculer l'erreur quadratique moyenne (RMSE)
rmse = mean_squared_error(y_test, predictions, squared=False)
print("Erreur quadratique moyenne (RMSE) :", rmse)

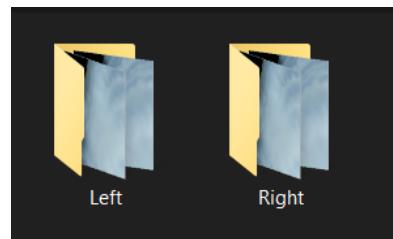
# Tracer le graphique des valeurs réelles par rapport aux valeurs prédites
plt.scatter(y_test, predictions)
plt.xlabel("Valeurs réelles")
plt.ylabel("Valeurs prédites")
plt.title("Valeurs réelles vs Valeurs prédites (Gradient Boosting Regressor)")
plt.show()

# saisir la taille et le poids du poisson
taille_poisson = 31
poids_poisson = 388
# Créer un tableau numpy avec la taille et le poids du poisson
nouveau_poisson = np.array([[taille_poisson, poids_poisson]])
# Utiliser le modèle pour prédire l'âge du poisson
age_pred = model.predict(nouveau_poisson)
# Afficher l'estimation de l'âge du poisson
print("L'estimation de l'âge du poisson est :", age_pred[0].round(0))
```

Détection et Séparation des Otolithes Droits et Gauches

Dans cette partie de l'étude, nous avons abordé le défi de diviser les images contenant deux otolithes, l'un droit et l'autre gauche, afin de faciliter leur analyse ultérieure. Pour ce faire, nous avons utilisé le langage de programmation **R** et la bibliothèque **EBImage** pour le traitement d'images.

```
Script R > Diveser_Otolithe_Groupe.R
1  library("EBImage")
2
3  # Chemin vers le dossier contenant les images
4  input_folder <- "C:/Users/ATLAS PRO ELECTRO/Desktop/plaice_right_png/plaice_right_png/14"
5  # Création des dossiers de sortie s'ils n'existent pas déjà
6  output_folder_right <- "C:/Users/ATLAS PRO ELECTRO/Desktop/images divesee/Right/"
7  output_folder_left <- "C:/Users/ATLAS PRO ELECTRO/Desktop/images divesee/Left/"
8  # Liste de fichiers dans le dossier d'entrée
9  image_files <- list.files(input_folder, full.names = TRUE)
10
11 # Taille de la zone à extraire
12 d <- 512
13 # Boucle à travers chaque image
14 for (file in image_files) {
15  # Lire l'image
16  img <- readImage[file]
17  # Déterminer les dimensions de l'image
18  h <- dim(img)[1]
19  w <- dim(img)[2]
20  # Coordonnées du centre de l'image
21  y <- w / 2
22  x <- h / 2
23  # Coordonnées du coin supérieur gauche de la zone à extraire
24  x1 <- max(1, x - d/2) # Assure que l'indice ne soit pas en dehors des limites de l'image
25  y1 <- max(1, y - d/2) # Assure que l'indice ne soit pas en dehors des limites de l'image
26  # Extraire la première moitié de l'image
27  img_left <- img[0:x, 0:w, ]
28  # Extraire la deuxième moitié de l'image
29  img_right <- img[x1:h, 0:w, ]
30  # Enregistrer les moitiés gauche et droite dans les dossiers respectifs
31  file_name <- basename(file)
32  output_file_left <- paste0(output_folder_left, file_name)
33  output_file_right <- paste0(output_folder_right, file_name)
34
35  writeImage(img_left, output_file_left)
36  writeImage(img_right, output_file_right)
```



dossier des otolithe droite :



