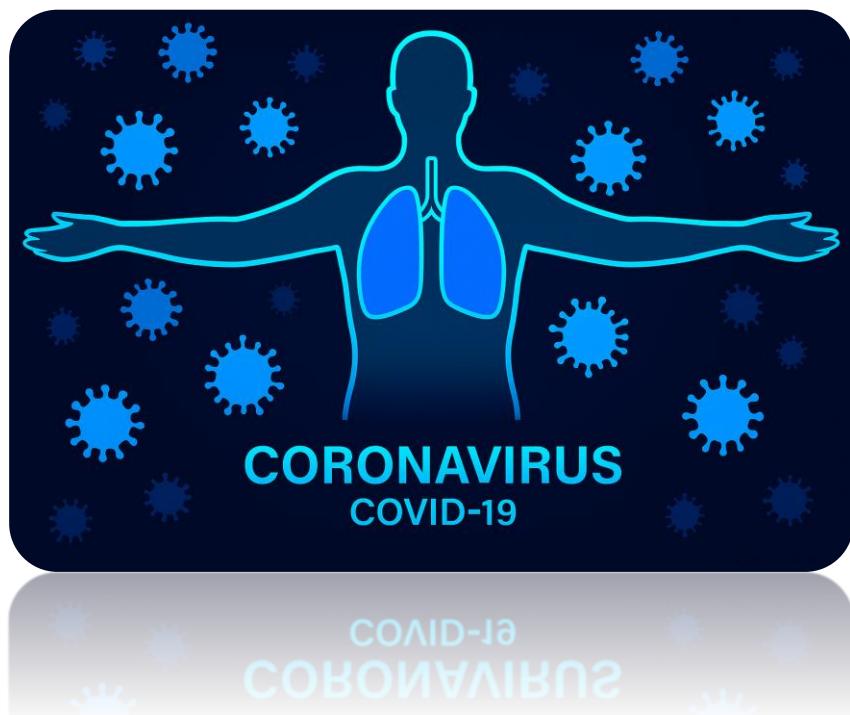




DataScientest

Analyse de Radiographies pulmonaires Covid-19



Présenté par :

- Cirine Bouamrane
- Léna Bacot
- Steven Moire
- Rafael Cepa

Encadré par :

Nicolas Mormiche

Sommaire :

Introduction :.....	4
Etape 1 : Exploration des données et DataViz'	4
Base de données :	4
Dataviz' :	6
Interprétation :	8
Steven Visualization :	9
🧠 Analyse métier	10
🛠 Recommandations techniques.....	10
Steven Template :.....	11
Stream lit Graph.....	12
Introduction :	12
Outils Collaboratifs :	12
Inspection des Fichiers Features :.....	12
Inspection détaillée des fonctions dans Features :.....	13
Analyse Complète (Déséquilibrée) :	14
Analyse Equilibrée :	18
Génération images masquées + Statistiques :	22
Exemple Génération Image Masqué :	22
Arborescence Automatique :	23
Nombre Pixels Total :	21
Nombre Pixels Affichés :	22
Luminosité Globale :	23
Luminosité Images Masquée :.....	23
Contraste Globale :	24
Contraste Images Masquées :	24
• Redimensionnement et conversion en niveaux de gris :	24
• Normalisation des pixels :.....	25
• Steven Augmentation data	26
select_folders.py.....	26
📁 Classe Python : Sélecteur de dossiers pour Jupyter Notebook	26
🔧 Fonctionnalités :	26

❖ Avantages par rapport à input() :	26
❖ Utilisation :	26
under_sampling.ipynb	28
▼ Fonction Python :	28
Application d'undersampling sur des images	28
🔧 Paramètres :	28
📋 Fonctionnement :	28
📤 Retour :	28
💡 Cas d'usage :	28
Main.ipynb	30
⌚ Script principal	30
🔧 Fonctionnement global :	30
Sélection interactive des dossiers sources :	30
Application de l'undersampling :	30
Validation visuelle :	30
📁 Gestion des chemins :	31
📊 Visualisation :	31
🎯 Cas d'usage :	31

Introduction :

Afin d'identifier plus facilement les cas positifs de Covid-19, l'analyse automatisée des radiographies pulmonaires apparaît comme une alternative précieuse, notamment lorsque les tests conventionnels ne sont pas disponibles. L'utilisation du deep learning appliquée à ces images a déjà montré des résultats prometteurs pour détecter les infections à Covid-19, parfois avec une précision supérieure à celle de radiologues expérimentés.

L'objectif de notre projet est donc de concevoir un système de classification binaire permettant de distinguer entre patients atteints et non atteints du Covid-19 à partir de radiographies thoraciques.

Etape 1 : Exploration des données et DataViz'

Base de données :

La base de données utilisée pour ce projet est la COVID-19 Radiography Database accessible sur Kaggle. Cette base de données regroupe différentes catégories d'images radiographiques pulmonaires permettant la détection et la classification des infections, notamment du Covid-19. Elle inclut :

- Des radiographies thoraciques de patients confirmés positifs au Covid-19.
- Des images de cas jugés normaux (sans infection pulmonaire détectée).
- Des radiographies de pneumonies virales autres que Covid-19.
- Des images présentant des opacités pulmonaires (infections pulmonaires non-COVID de type "Lung Opacity").

Les tailles des sous-ensembles composant la base de données sont représentées dans le tableau suivant :

<i>Classes</i>	<i>Images</i>	<i>Masks</i>
<i>Covid</i>	3616	3616
<i>Normal</i>	10192	10192
<i>Viral_pneumonia</i>	1345	1345
<i>Lung_opacity</i>	6012	6012

Ces images proviennent de multiples sources publiques, publications spécialisées et collaborations avec des hôpitaux.

Un exemple d'une image de la classe normale et son mask sont représentées dans la figure ci-dessous (Figure01).



Figure 01 : Exemple d'une image de la base de données et son mask (classe normal)

Dataviz' :

Histogramme de la luminosité et du contraste :

- **Luminosité :**

Il s'agit de l'intensité lumineuse globale d'une image, c'est-à-dire du niveau moyen de clarté ou d'obscurité perçu sur l'ensemble de l'image. Mathématiquement, c'est simplement la moyenne des valeurs des canaux de chaque pixel. Pour une image en couleurs (RGB), on utilise la pondération de la norme ITU-R BT.601, correspondant à la sensibilité de l'œil humain (0.299 pour R, 0.587 pour G, 0.114 pour B).

- **Contraste :**

Le contraste représente la différence de luminosité entre les zones claires et sombres d'une image. Mathématiquement, c'est l'écart-type (std) de la valeur des pixels. Un contraste élevé signifie que la différence entre les parties les plus sombres et les plus claires est importante ; à l'inverse, un faible contraste indique que les valeurs de gris sont proches les unes des autres, rendant les détails moins visibles.

Un histogramme de luminosité et de contraste fournit des informations essentielles sur la répartition des tons et la qualité d'exposition d'une image, l'histogramme permet de juger de la bonne exposition, du niveau de contraste, de repérer d'éventuelles pertes d'information, et de guider les ajustements à apporter pour optimiser une image.

Il indique comment les pixels de l'image sont répartis des tons les plus foncés (gauche) aux plus clairs (droite) :

- À gauche : pixels sombres (ombres, noirs)
- Au centre : tons moyens (gris)
- À droite : pixels clairs (hautes lumières, blancs)

Il permet d'identifier :

- Une sous-exposition (pic vers la gauche : image trop sombre)
- Une surexposition (pic vers la droite : image trop claire)
- Un bon contraste (répartition étalée de la gauche à la droite : l'image contient à la fois des zones sombres et claires, donc beaucoup de détails)
- Un faible contraste (histogramme ramené vers le centre, l'image paraît "plate" avec surtout des tons moyens)

L'histogramme aide aussi à :

- Éviter l'écrêtage : apparition de barres hautes à l'extrême gauche ou droite, signalant une perte d'information dans les ombres ou les hautes lumières.
- Ajuster la correction de la luminosité et du contraste pour améliorer la lisibilité ou la qualité technique de l'image.

Nous avons généré les histogrammes de luminosité pour chaque catégorie d'images (normal, Covid, pneumonie et lung). Les résultats correspondants sont illustrés dans les figures suivantes :

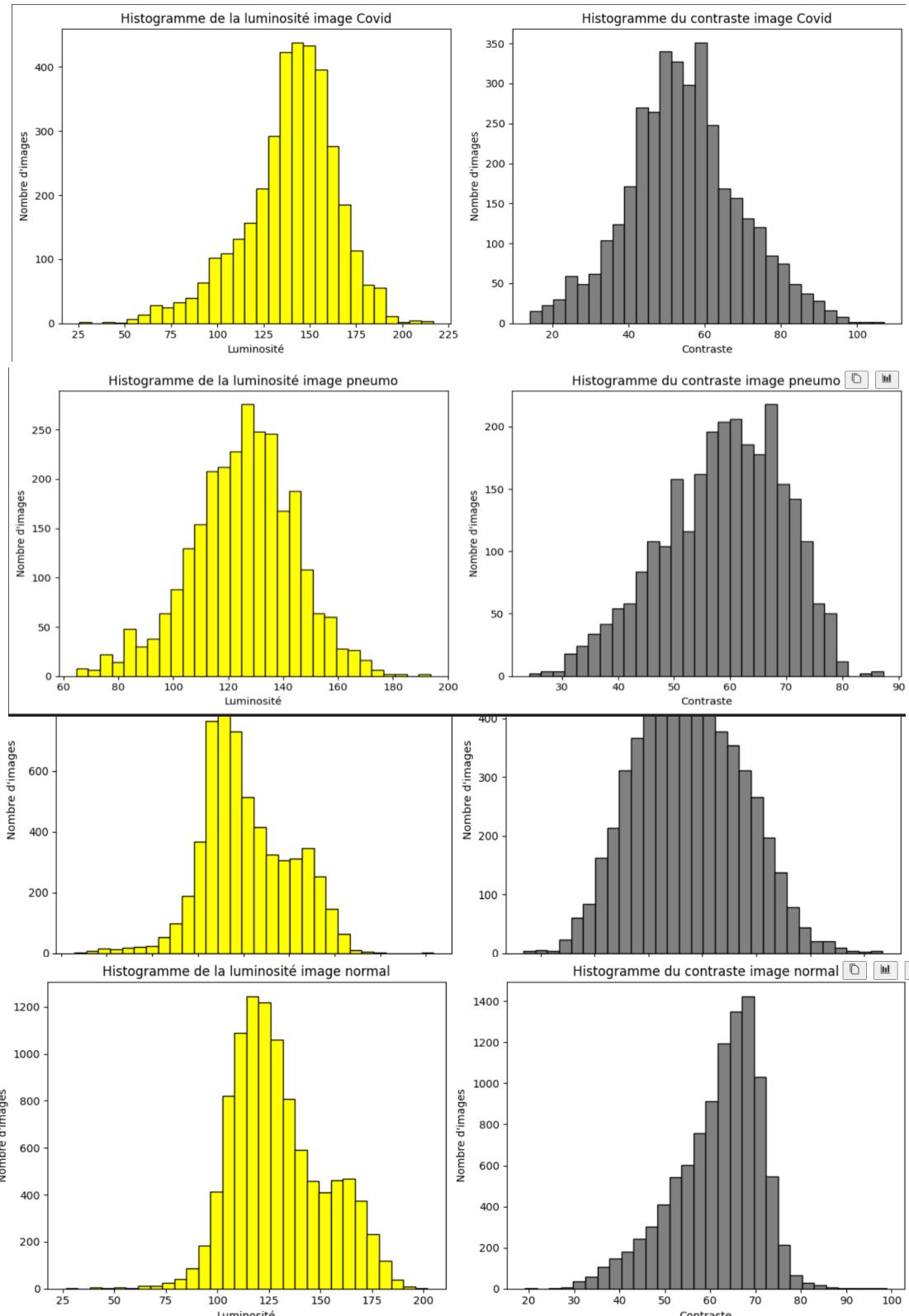


Figure 02 : Histogramme de la luminosité et du contraste des différentes images de la dataset

Interprétation :

A savoir :

- **Axe horizontal :** les valeurs de gris, de 0 (noir) à 255 (blanc).
- **Axe vertical :** nombre d'images.

	Histogramme luminosité	Histogramme contraste
<i>Image normale</i>	Ton vers le clair	Faible contraste
<i>Image covid</i>	Ton vers le clair	Bon contraste
<i>Image Pneumo</i>	Ton moyen (gris équilibré)	Bon contraste
<i>Image lung</i>	Ton vers le clair	Bon contraste

Luminosité Globale :

Exprime la répartition de la luminosité des images, pour chaque catégorie.

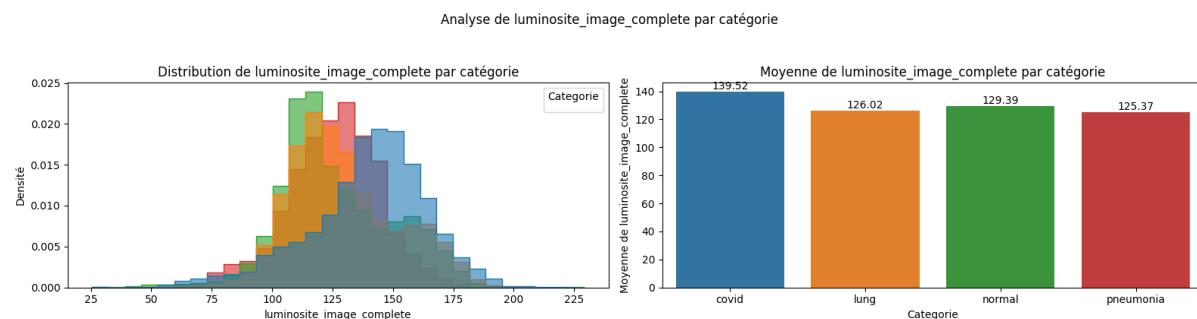


Figure 03 : Distribution Luminosité Globale

Contraste Globale :

Exprime la répartition du contraste des images, pour chaque catégorie.

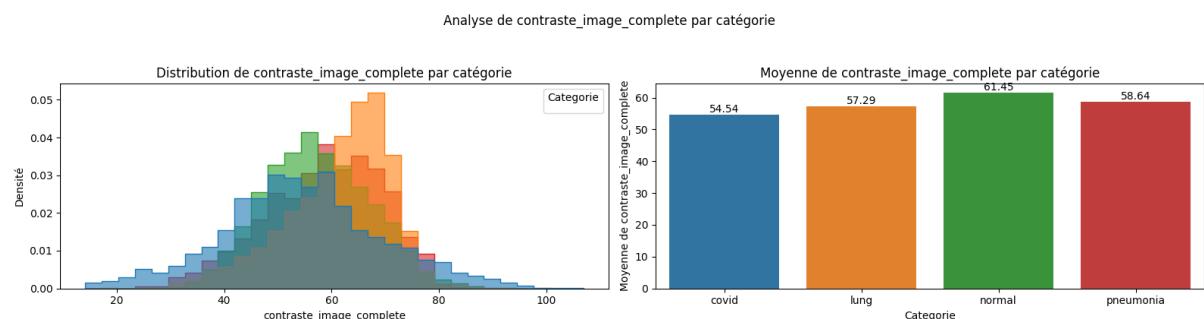


Figure 04 : Distribution du contraste Globale

Steven Visualization :

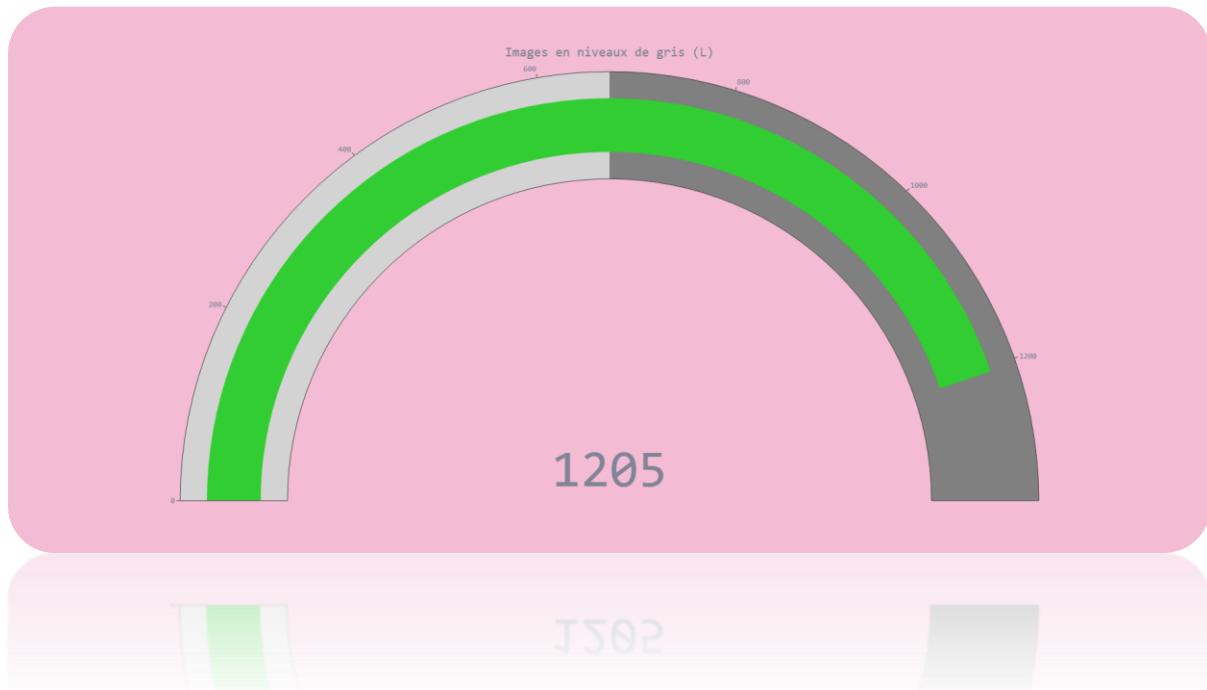


Figure 03 : Visualisation des images en L et en RGB de la classe Viral Pneumonia

⚠ 140 images RGB détectées (hors norme) sur un total de 1345 images provenant de Viral Pneumonia.

Après analyse, nous avons conclu qu'il s'agit en fait de 'faux RGB', c'est à dire des images encodées en 3 canaux, mais représentant en réalité une image L. En effet, quand on a essayé d'afficher quelques intrus, nous avions l'impression de voir des niveaux de gris. Pour chaque intrus 'img', convertie en np.array, nous avons :

```
img_array[:, :, 0] == img_array[:, :, 1] == img_array[:, :, 2]
```

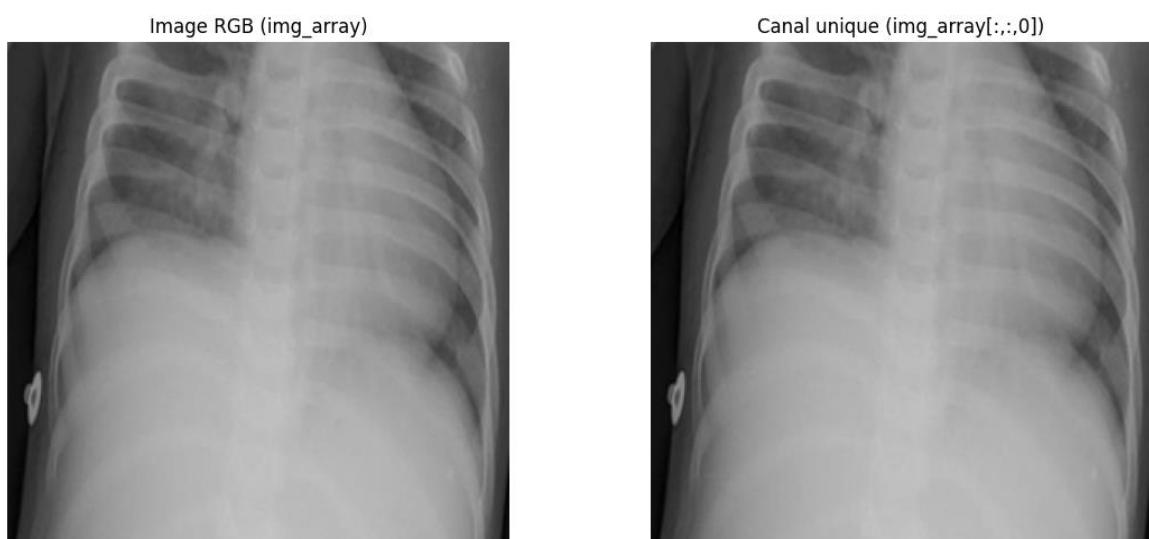


Figure 04 : exemple d'images en RGB de la classe Viral Pneumonia

Ces 140 images sont bien codées en 3 canaux RGB, mais on a, à chaque fois, la même valeur pour les trois canaux, c'est donc une redondance d'informations, et la conversion en L sera alors encore plus simple. Pour rappel, une image RGB est convertie en L selon la norme ITU-R BT.601 (citée plus haut), grâce à la formule :

$$L = 0.299R + 0.587G + 0.114B.$$

Où :

- L : valeur du pixel en nuance de gris résultante de la conversion
- R, G, B : valeurs du pixel en RGB que l'on veut convertir

Il est donc évident ici, qu'avec $R = G = B$:

$$L = 0.299R + 0.587R + 0.114R \leftrightarrow L = R = G = B$$

Autrement dit, le pixel converti en L n'est autre que la même valeur commune aux trois channels, on peut donc simplement choisir le premier canal (R, i.e. `img_array[:, :, 0]`) pour ‘convertir’ l'image en L.

Nous avons remarqué qu'en réalité, ce problème est exactement le même pour les masks, qui sont tous des “faux RGB”. Pour la suite de notre projet, nous avons décidé d'utiliser les fonctions de conversion des packages reconnus (PIL, cv2 ...), même si utiliser le premier canal (par exemple) suffirait. En effet, nous voulons tester différentes approches, et essayer de comprendre ce qu'apportent ces fonctions. Il nous semblait cependant important de montrer notre réflexion face à ce problème.  **Analyse métier**

- Les radiographies doivent être en niveau de gris (L), car les informations pertinentes ne sont pas dans les couleurs.
- La présence de plusieurs images en RGB est donc anormale.
- Cela peut indiquer des erreurs de traitement ou d'export depuis un outil d'annotation.



Recommandations techniques

- Identifier les images en RGB et les vérifier visuellement.
- Si ce sont bien des radios, les convertir en L via `img.convert("L")` pour :
 - Homogénéiser les données
 - Réduire la taille mémoire
- Ces anomalies sont visibles dans le rapport template d'exploration automatique généré (voir lien juste après).

Steven Template :

<https://1drv.ms/x/c/9e322a9006fb4eb1/ERjCTdYjnJBFhoRxeNj00EBlyoYCRADMprJFlOrgObT0Q?e=idyaqO>

Stream lit Graph

Introduction :

La bibliothèque StreamLit sur python, nous permet de créer des Dashboard interactifs.

Ceux-ci ont été conçus afin de nous permettre de créer des outils afin de faciliter notre collaboration.

Le Stream Lit peut appeler les fonctions en backend pour effectuer certaines tâches dans notre data-workflow et nous présenter des visualisations d'analyses que l'on a effectué.

A la suite, vous trouverez un aperçu de ces fonctionnalités et un cas d'application pratique sur notre problématique.

Outils Collaboratifs :

Inspection des Fichiers Features :

Voici un aperçu de la fenêtre d'inspection des fichiers.

Analyse du dossier Features							
Dossier analysé: C:\Users\Léna\Documents\A_Repos\DS_COVID\src\features			Dossier trouvé				
14 fichier(s) Python trouvé(s)							
Fichier	Dossier	Taille (KB)	Lignes	Modifié	Statut	PEP8	
0 Augmentation/Augmentation.py	Augmentation	0.25	8	2025-07-23 16:56:20	OK	✗ (2)	
1 Inspector/Features_Core.py	Inspector	9.93	229	2025-07-23 15:14:43	OK	✗ (39)	
2 Inspector/Features_Frontend.py	Inspector	15.69	345	2025-07-23 15:13:19	OK	✗ (95)	
3 Inspector/Features_Inspect_Backend.py	Inspector	21.6	521	2025-07-23 15:32:24	OK	✗ (119)	
4 Traitement_Data/C_traitement.py	Traitement_Data	4.94	141	2025-07-24 20:06:53	OK	✗ (17)	
5 Traitement_Data/Traitement_copy.py	Traitement_Data	9.61	248	2025-07-23 19:40:46	OK	✗ (50)	
6 Traitement_Data/Traitement.py	Traitement_Data	9.61	248	2025-07-23 19:40:46	OK	✗ (50)	
7 Traitement_Data/traitement_modulaire.py	Traitement_Data	9.63	243	2025-07-24 21:56:35	OK	✗ (58)	
8 Traitement_Data/workflow_copy.py	Traitement_Data	0.63	19	2025-07-23 20:03:27	Erreur code	✗ (5)	
9 Traitement_Data/workflow.py	Traitement_Data	0.69	23	2025-07-24 19:41:21	Erreur code	✗ (3)	

Figure 05 : Inspection Fichiers Python Backend

Cet affichage nous permet de voir l'évolution des fichiers de fonctions backend d'un simple coup d'œil !

On peut donc voir le nombre de lignes et la validation PEP8, nous permettant de revoir les fichiers adéquats et donc de gagner en organisation.

Inspection détaillée des fonctions dans Features :

Pour chacun des fichiers détectés précédemment, on vient scanner toutes les fonctions (à l'aide de la librairie Inspect) afin de les afficher.

Ceci sert pour le travail collaboratif afin de rester informé de l'existence des fonctions créées ou mises à jour par nos équipes.

The screenshot shows a web-based interface for inspecting Python functions. At the top, it says "53 fonction(s) trouvée(s) / 53". Below is a table with columns: Fonction, Dossier, Fichier, Lignes, Paramètres, Retour, Doc, and Description. The table lists various functions like augmentation_data_test, cat, analyze_module_functions, etc., with their respective details. Below the table is an "Index de la fonction à détailler" section with a dropdown set to "0". At the bottom, a panel titled "Détails de la fonction : augmentation_data_test" shows the function definition: `def augmentation_data_test():`. The entire interface has a dark theme with light-colored text and tables.

Figure 06: Inspection Fonctions Python Backend

Il est ensuite possible d'afficher un panel détaillé avec toutes les informations essentielles de la fonction !

Ceci incluant le nom de la fonction, la doc string, le nombre de lignes, les arguments et la valeur de return.

This screenshot shows a detailed view of the `list_image_files` function. It includes the function definition (`def list_image_files(image_folder)`), file information ("Fichier: traitement_modulaire.py"), line count (8), parameter count (1), and return value (images_list). A description below states "Retourne la liste des fichiers images dans un dossier.". Below this is a "Paramètres détaillés" section with a single parameter: `image_folder : None (Défaut:None)`. There is also a link to "Afficher le code source". The bottom part shows the function's implementation in Python, which lists files in a folder if they have specific extensions (.png, .jpg, .jpeg).

Figure 07 : Détail Fonctions Python Backend

Analyse Complète (Déséquilibrée) :

Nous allons utiliser les fonctionnalités de l'interface afin de traiter et analyser nos données d'images.

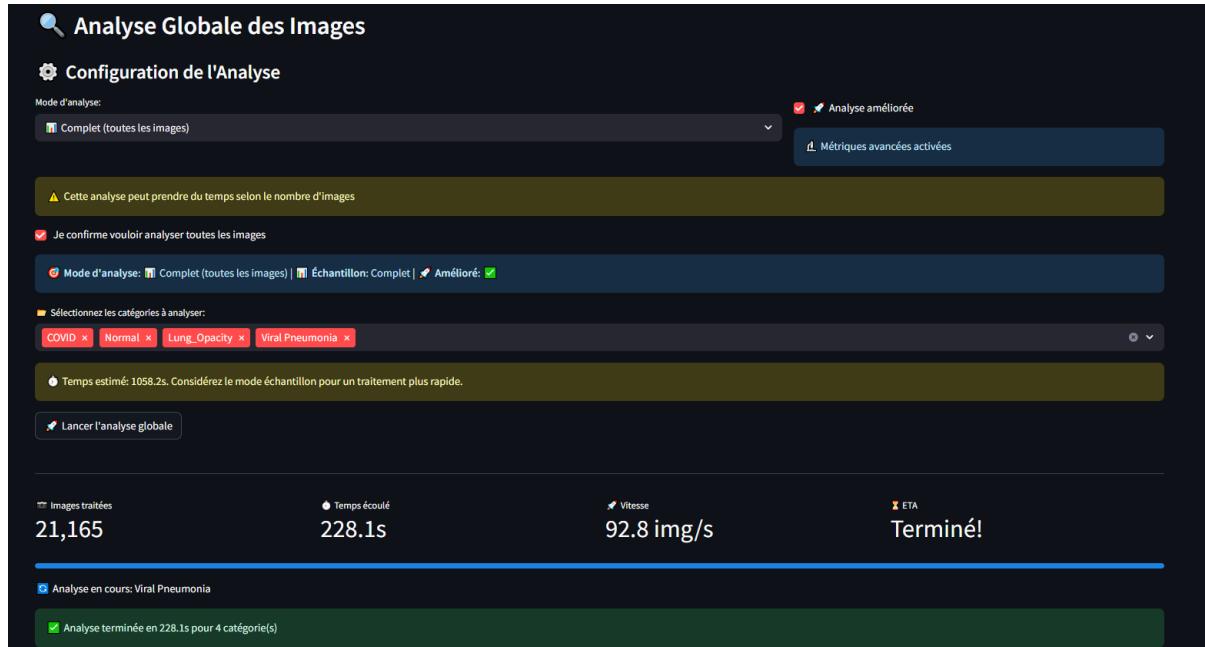


Figure 08 : Menu Principal, sélection des données

On lance une analyse sur l'ensemble des images.

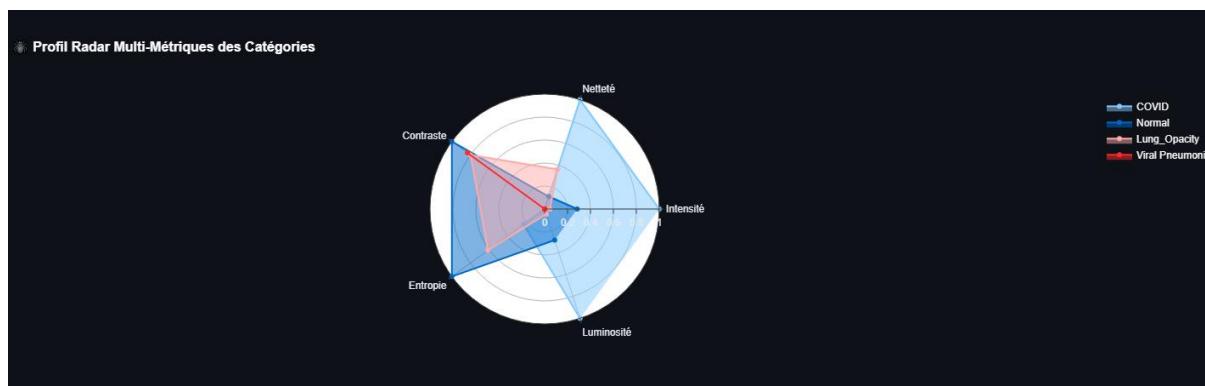


Figure 09 : Radar Chart, Données/Métriques par Catégories

On peut déjà noter visuellement des domaines différents pour chaque catégorie.

Luminosité / Intensité : Moyenne des valeurs de pixels, reflète la clarté globale de l'image.

Contraste : Écart-type des valeurs de pixels, mesure la variation autour de la moyenne.

Netteté : Quantifie la présence de détails et de contours, via la fonction Laplacien.

Entropie : Mesure la diversité des intensités, indique la richesse d'information.



Figure 10 : Test Anova, différence entre Catégories

Et l'on confirme ce résultat par un test ANOVA.

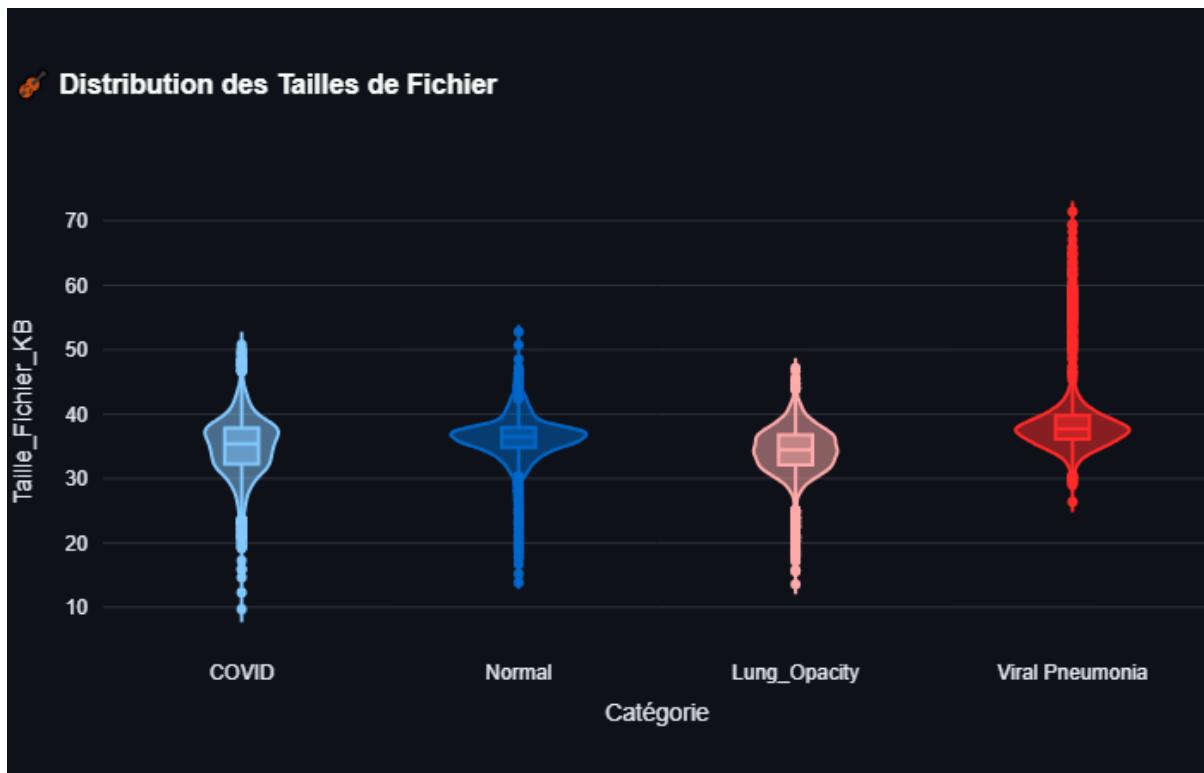


Figure 11 : Taille des fichiers par catégorie

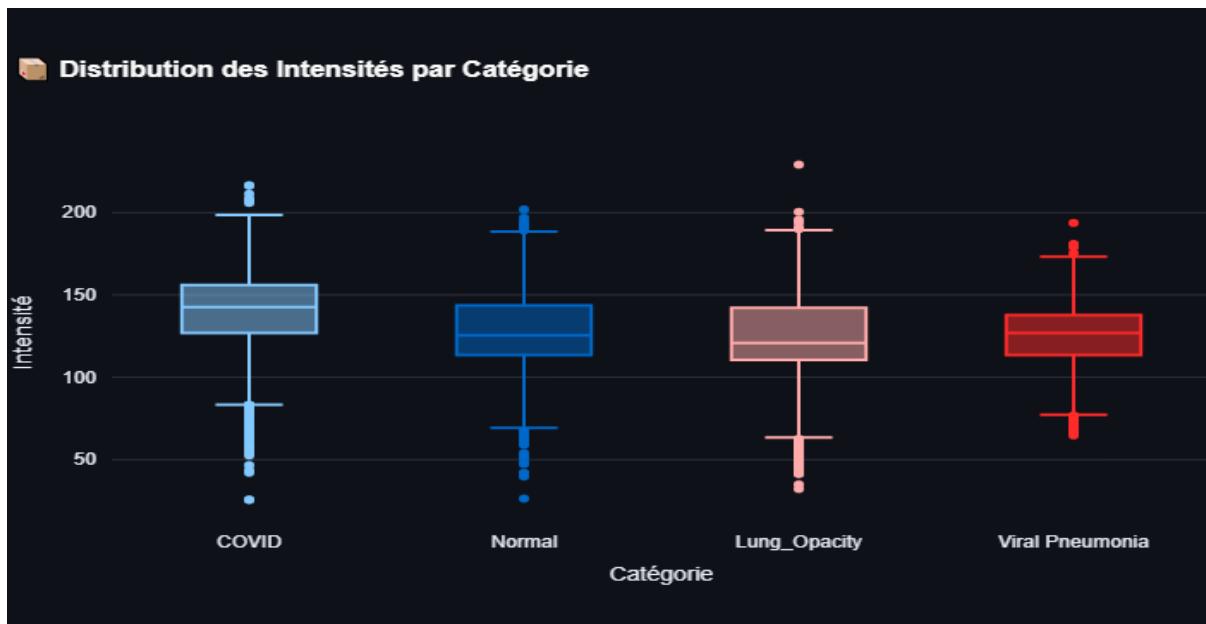


Figure 12 : Intensité par Catégorie

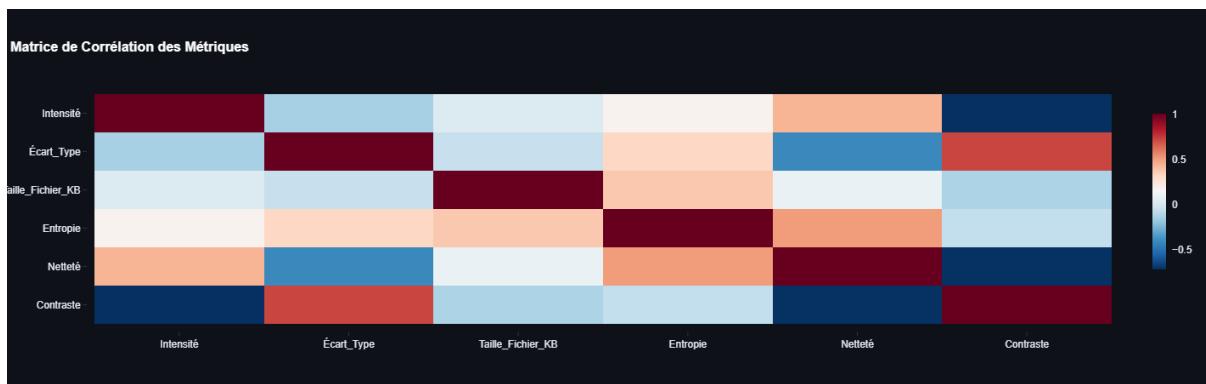


Figure 13 : Corrélation des métriques

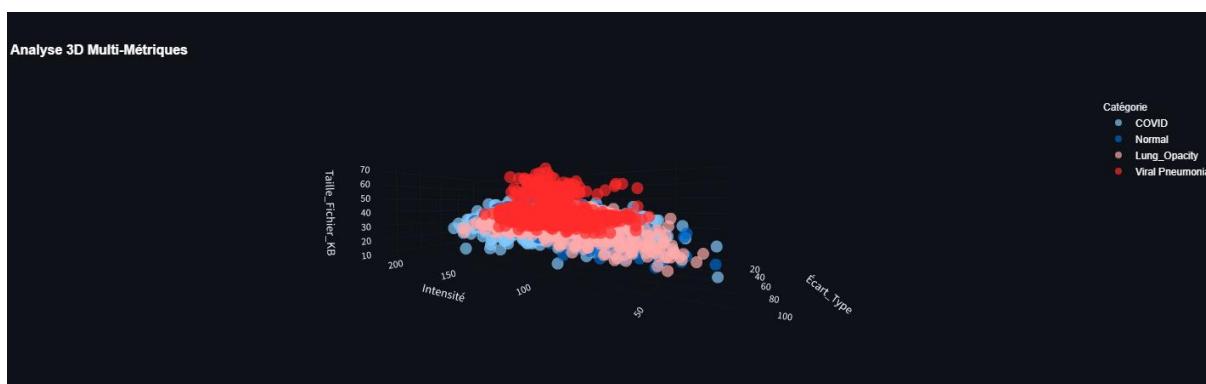


Figure 14 : Répartition des données dans un espace 3D

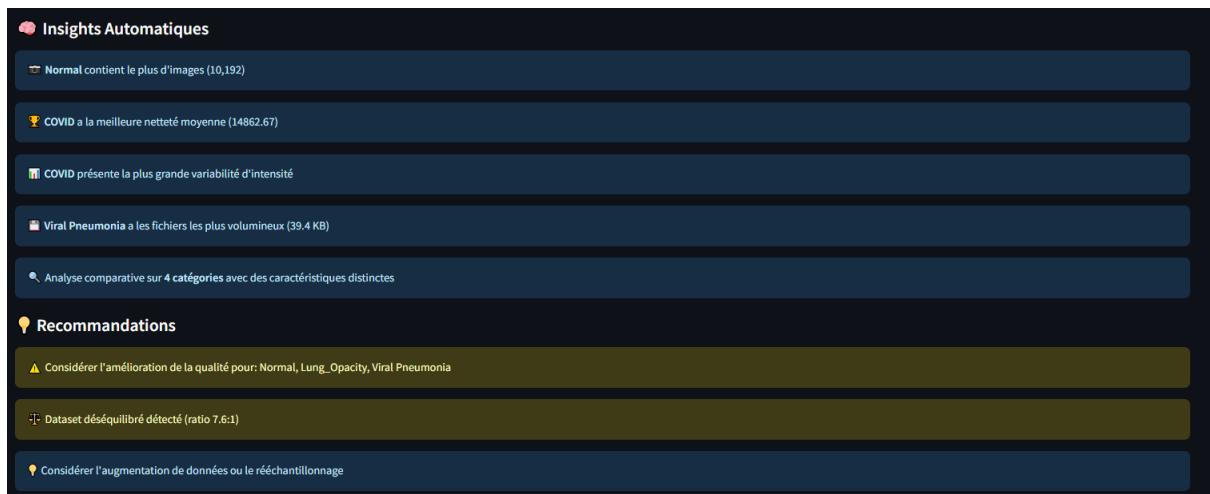


Figure 15 : Recommandations Automatiques

Analyse Equibrée :

Suite à cette analyse, on se rend compte que les données ne sont pas équilibrées, on va donc retenter l'analyse avec un échantillon de 1000 pour chacune des catégories et voir les différences sur nos indicateurs.

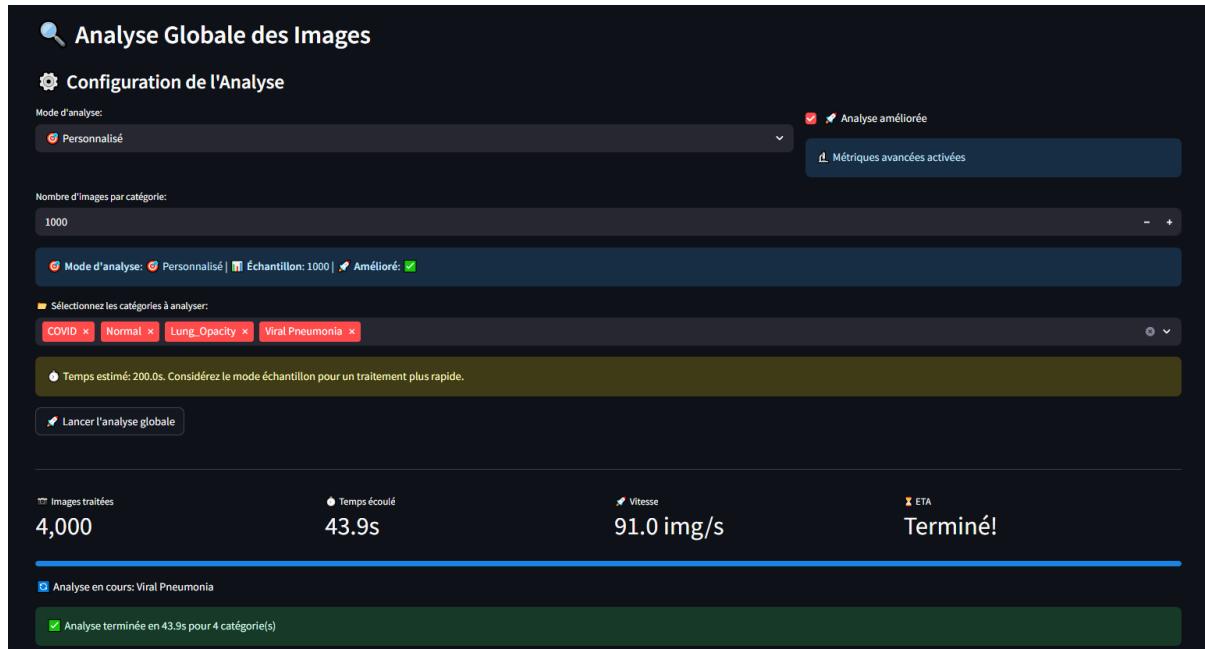


Figure 16 : Menu Principal, sélection des données (équilibrés)

Analyse Comparative Complète												
Résumé Comparatif												
Catégorie	Nombre_Images	Largeur_Moyenne	Hauteur_Moyenne	Intensité_Moyenne	Écart_Type_Intensité	Taille_Fichier_Moyenne_KB	Entropie_Moyenne	Netteté_Moyenne	Contraste_Moyen	Densité_Contours	Luminosité_Moyenne	
0 COVID	1000	299	299	143.033	24.4117	35.4991	7.2011	14858.6255	0.367	0	0.5609	
1 Normal	1000	299	299	127.4651	20.4212	38.1959	7.4433	14657.2898	0.4958	0	0.4999	
2 Lung_Opacity	1000	299	299	125.9232	23.3011	34.4964	7.3877	14823.8537	0.4698	0	0.4938	
3 Viral Pneumonia	1000	299	299	125.8511	19.2322	39.4044	7.2261	14770.4381	0.4782	0	0.4935	

Métriques Clés

Total Images 4,000	Résolution Moy. 89.4K px	Taille Moy. 36.9 KB	Meilleure Netteté COVID
-----------------------	-----------------------------	------------------------	----------------------------

Figure 17 : Menu Principal, Aperçu des données (équilibrés)

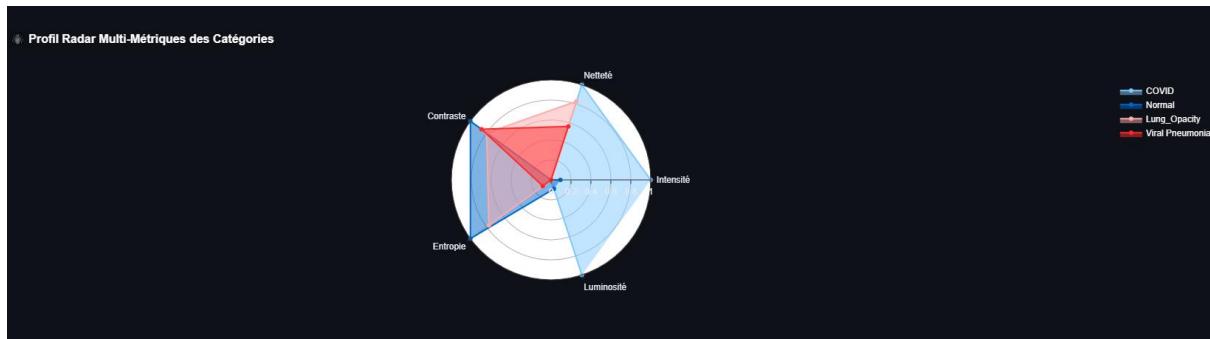


Figure 18 : Menu Principal, sélection des données (équilibrés)

On peut noter sur cet ensemble plus équilibré que la représentation des domaines est différente par rapport à l'analyse précédente.

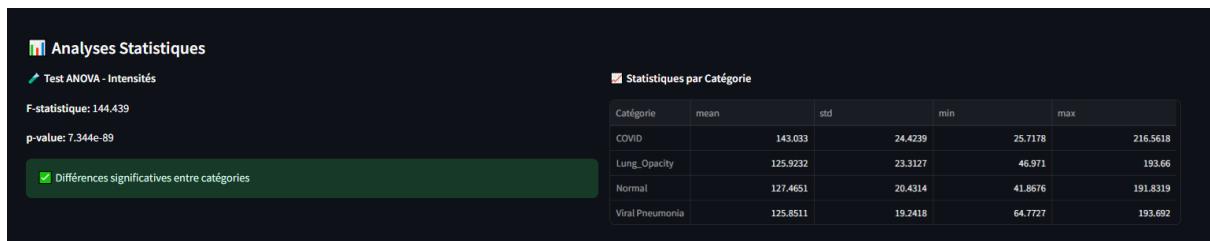


Figure 19 : Test Anova, Différences Catégories (équilibrés)

Les différences sont toujours significatives.

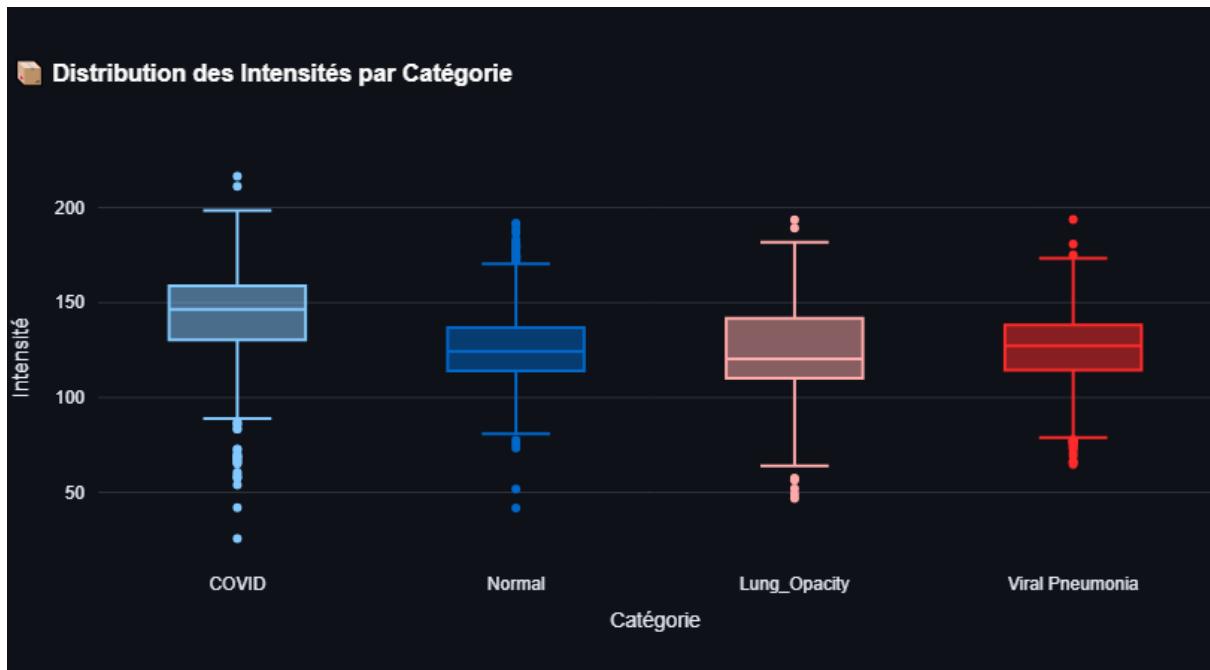


Figure 20 : Intensité par Catégories (équilibrés)



Figure 21 : Taille de fichiers par catégorie (équilibrés)

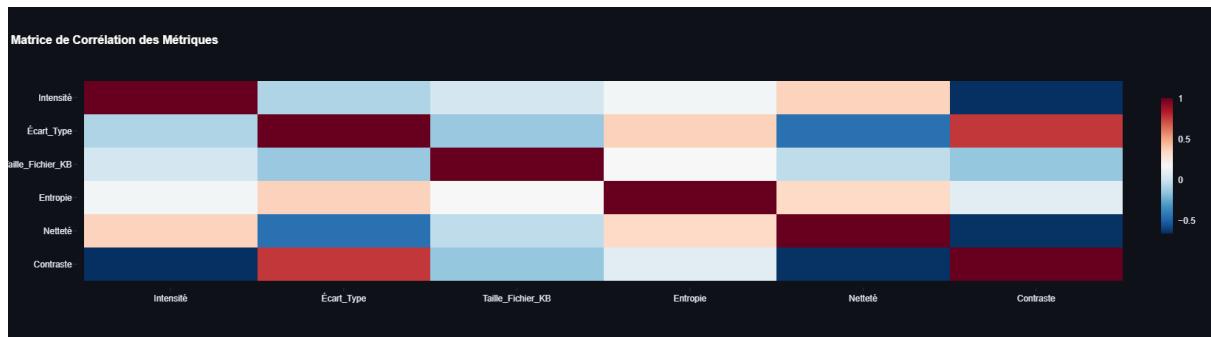


Figure 22 : Corrélation des métriques (équilibrés)



Figure 23 : Recommandations Automatiques (équilibrés)

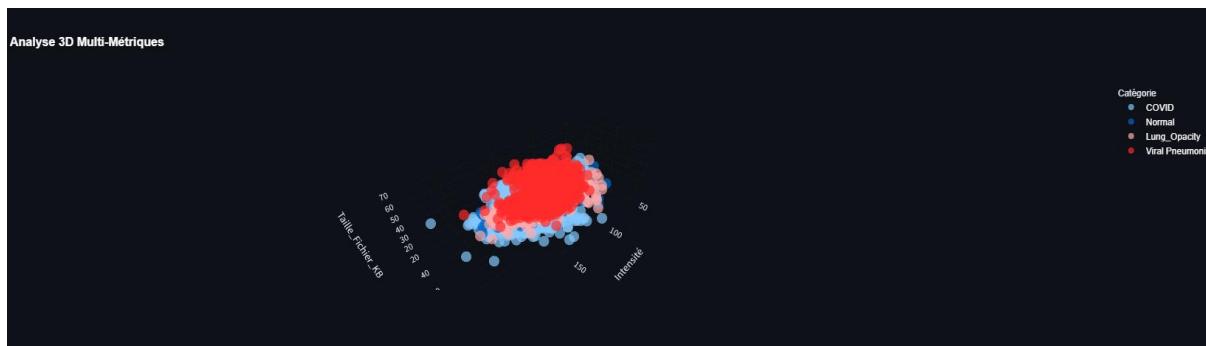


Figure 24 : Représentation des données dans un espace 3D (équilibrés)

Génération images masquées + Statistiques :

Cette fonctionnalité appelée par un notebook, permet de générer des images masquées à l'aide d'images et de masques (nos datas).

Pendant cette opération, des statistiques sont également évalués et représentées.

Voici les statistiques du masquage sur l'ensemble des données, avec un paramètre :

dsize = (256x256), ce qui correspond à une image de 256x256 pixels.

Exemple Génération Image Masqué :

La fonction permet en lui donnant une image et un masque, de l'appliquer pour générer une image masquée, avec une possibilité de redimensionnement de l'image.



Figure 25 : Exemple, Masquage des images

Arborescence Automatique :

Afin de pouvoir garder un environnement propre, une arborescence automatique pour trier les éléments générés a été mise en place.

Celle-ci génère dans le dossier "processed":

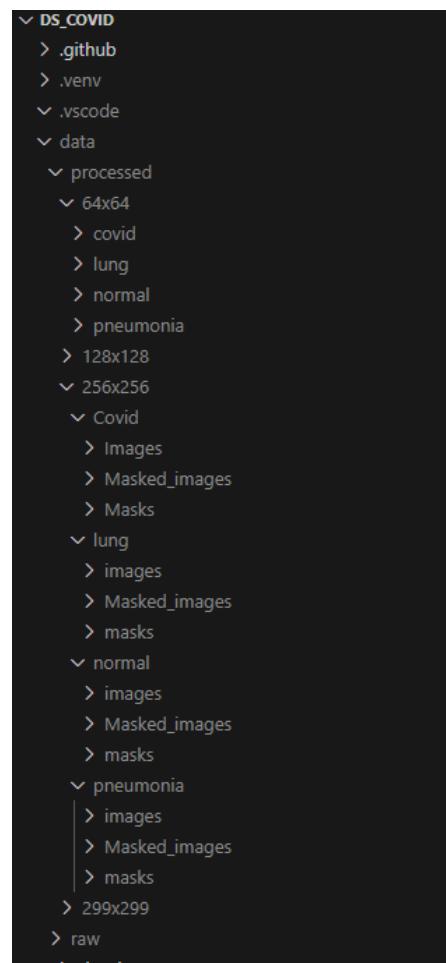
Un premier dossier en fonction de la résolution,
Exemple : 256x256 (Nom du dossier).

Ensuite, dans ce dossier, 4 autres sont créées pour chacune des catégories :

- Covid
- Lung (Opacity)
- Normal
- Pneumonia

Dans chacun de ces dossiers, on va générer des dossiers par type d'éléments générés :

- Images redimensionnés
- Masques redimensionnés
- Images avec Masque appliqué



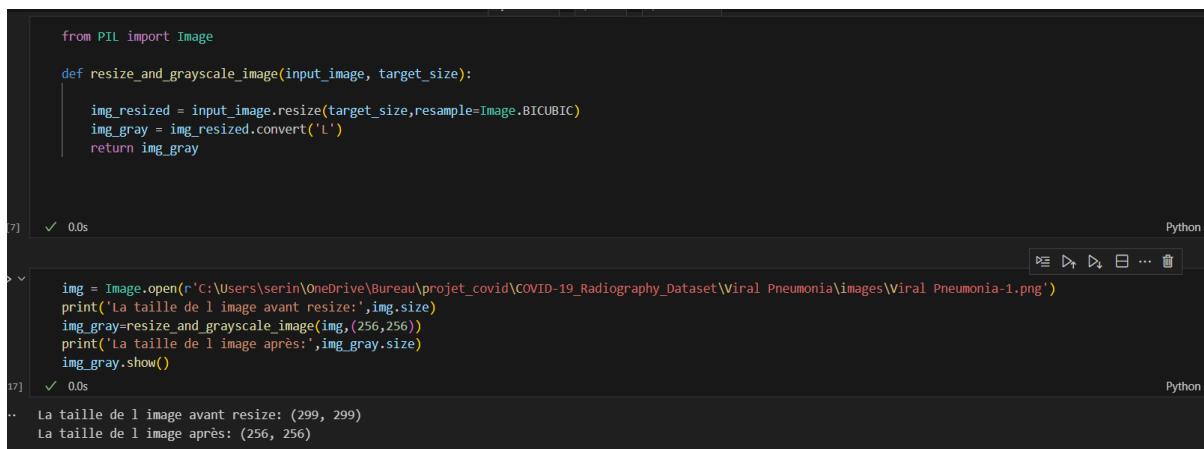
Etape 02: Pre-processing et features engineering:

- Redimensionnement et conversion en niveaux de gris :

Dans la phase initiale du pré-processing des données, nous appliquons un redimensionnement des images dont la dimension initiale est de 299 pixels, pour les réduire à une taille uniforme de 256 pixels. Ce choix, vise à garantir une cohérence des entrées du modèle tout en facilitant le traitement par lot et en réduisant la charge computationnelle. Ce redimensionnement est réalisé en utilisant des techniques d'interpolation adaptées, telles que l'interpolation bicubique, afin de préserver au mieux la qualité et les détails pertinents des images.

Par la suite, la conversion de l'ensemble des images du format RGB vers une représentation en niveaux de gris est effectuée. Cette transformation réduit la dimension d'entrée des données de trois canaux à un seul, diminuant ainsi la complexité du modèle tout en conservant l'essentiel de l'information structurelle pertinente, particulièrement dans le contexte d'images médicales où la couleur est souvent secondaire. Cette étape est particulièrement judicieuse dans un cadre de classification, car elle simplifie le signal d'entrée, réduit le bruit potentiel lié à la variation chromatique, et permet d'optimiser les performances des algorithmes d'apprentissage en concentrant l'analyse sur les contrastes et textures essentielles à la discrimination des classes.

Le code utilisé est dans la figure ci-dessous:(Figure)



```
from PIL import Image

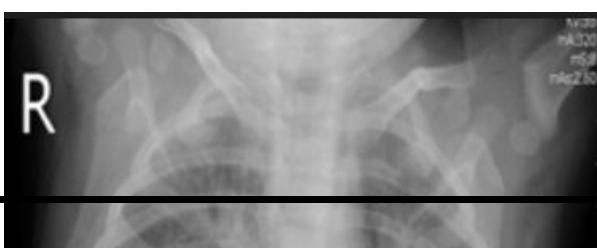
def resize_and_grayscale_image(input_image, target_size):
    img_resized = input_image.resize(target_size, resample=Image.BICUBIC)
    img_gray = img_resized.convert('L')
    return img_gray

img = Image.open(r'C:\Users\serin\OneDrive\Bureau\projet_covid\COVID-19_Radiography_Dataset\Viral Pneumonia\images\Viral Pneumonia-1.png')
print('La taille de l image avant resize:',img.size)
img_gray=resize_and_grayscale_image(img,(256,256))
print('La taille de l image après:',img_gray.size)
img_gray.show()

La taille de l image avant resize: (299, 299)
La taille de l image après: (256, 256)
```

Figure 33 : le code utilisé pour le redimensionnement et la conversion en niveau de gris

Un exemple des images de la classe pneumonie obtenue après conversion en niveau de gris et redimensionnement est représenté dans la figure (Figure) suivante :



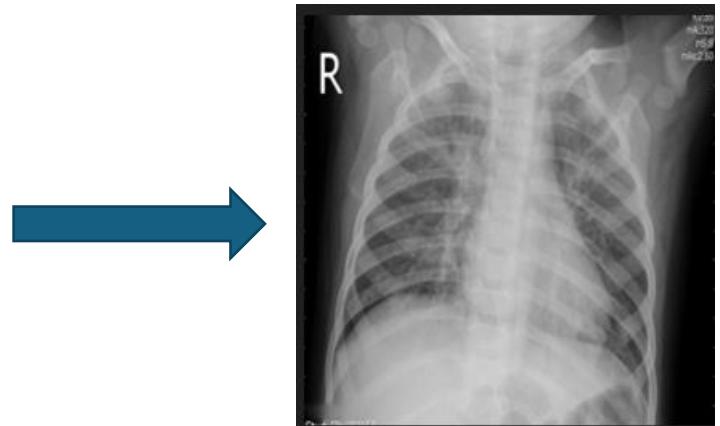


Figure 34 : Exemple d'une image de la classe pneumonie avant et après redimensionnement et conversion

- **Normalisation des pixels :**

Après redimensionnement, la normalisation des valeurs de pixels est essentielle pour aider les modèles d'apprentissage profond à converger rapidement, dans notre cas nous avons choisi de normaliser les pixels dans une plage [-1,1], le code utiliser est représenter dans la figure suivante : (Figure0)

```
def img_normalise(input_image):
    input_image=input_image.astype('float32')
    img_norm=(input_image/127.5)-1
    return img_norm
```

Figure 35 : Fonction de normalisation

- **Steven et Raphael Augmentation data**

`select_folders.py`

Classe Python : Sélecteur de dossiers pour Jupyter Notebook

Ce code définit une classe FolderSelector permettant de sélectionner interactivement des dossiers dans un notebook Jupyter.

Bien que Jupyter Notebook accepte les commandes `input()` (qui s'affichent en haut du notebook), cette méthode peut sembler contre-intuitive et peu ergonomique. Pour offrir une expérience utilisateur plus fluide, ce widget utilise les bibliothèques `ipywidgets` et `IPython.display` afin de proposer une interface graphique intégrée directement sous la cellule de code.

Fonctionnalités :

- Ajout de chemins de dossiers valides via un champ de texte
- Validation en temps réel de l'existence du dossier
- Affichage des dossiers sélectionnés avec feedback visuel
- Bouton "Terminer" pour valider la sélection et masquer automatiquement l'interface après 5 secondes

Avantages par rapport à `input()` :

- Interface intégrée au flux du notebook
- Feedback immédiat sur la validité des chemins
- Masquage automatique après validation
- Expérience utilisateur améliorée

Utilisation :

1. Instancier la classe : `selector = FolderSelector()`
2. Afficher le widget : `selector.afficher()`
3. Ajouter les dossiers via l'interface
4. Récupérer les chemins via `selector.chemins` une fois `selector.fini == True`

```
● ● ●

from ipywidgets import Text, VBox, Button, Output, Layout
from IPython.display import display
import os
import threading
import time

class FolderSelector:
    def __init__(self):
        self.chemins = []
        self.fini = False
        self.out = Output()

        self.input_box = Text(placeholder="Collez un chemin vers un dossier")
        self.bouton_valider = Button(description="Ajouter")
        self.bouton_terminer = Button(description="Terminer")

        self.bouton_valider.on_click(self.ajouter_dossier)
        self.bouton_terminer.on_click(self.terminer)

        self.ui = VBox([
            self.input_box,
            self.bouton_valider,
            self.bouton_terminer,
            self.out
        ])

    def afficher(self):
        display(self.ui)

    def ajouter_dossier(self, _):
        chemin = self.input_box.value.strip()
        if chemin == "":
            with self.out:
                print("⚠ Chemin vide ignoré.")
            return
        if os.path.isdir(chemin):
            self.chemins.append(chemin)
            with self.out:
                print(f"✓ Ajouté : {chemin}")
        else:
            with self.out:
                print(f"✗ Invalide : {chemin}")
        self.input_box.value = ""

    def terminer(self, _):
        # Désactive tous les widgets
        self.bouton_valider.disabled = True
        self.bouton_terminer.disabled = True
        self.input_box.disabled = True

        self.fini = True
        with self.out:
            print("\n✓ Saisie terminée.")
            print("■ Dossiers retenus :")
            for path in self.chemins:
                print(f" - {path}")

    # Lance un timer pour cacher le widget après 5 secondes
    def cacher_ui():
        time.sleep(5)
        self.ui.layout.display = 'none'

    threading.Thread(target=cacher_ui, daemon=True).start()
```

Figure 36 : Code de select_folders.py

under_sampling.ipynb

Fonction Python :

Application d'undersampling sur des images

Cette fonction apply_undersampling permet de réduire le nombre d'images dans un dossier en sélectionnant aléatoirement un nombre défini d'images. Cette technique, appelée undersampling, est couramment utilisée en machine learning pour équilibrer des jeux de données déséquilibrés.

Paramètres :

data_dir : chemin du dossier contenant les images sources

target_count : nombre d'images souhaité en sortie

output_dir : dossier de destination des images sélectionnées

Fonctionnement :

Liste toutes les images .png du dossier source, si le nombre d'images dépasse target_count, en sélectionne aléatoirement

target_count Copie les images sélectionnées dans le dossier de sortie, crée le dossier de sortie s'il n'existe pas

Retour :

Liste des logs détaillant les opérations effectuées

Cas d'usage :

Équilibrage de classes dans un dataset d'entraînement

Réduction de la taille d'un dataset trop volumineux

Préparation de données pour l'apprentissage automatique

```
import os
import random
import shutil
from glob import glob

def apply_undersampling(data_dir, target_count, output_dir):
    import os, random, shutil
    from glob import glob

    logs = []
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    images = glob(os.path.join(data_dir, "*.png"))
    logs.append(f"Recherche dans : {data_dir}")
    logs.append(f"Images trouvées : {len(images)}")

    if len(images) > target_count:
        selected_images = random.sample(images, target_count)
    else:
        selected_images = images

    for img_path in selected_images:
        shutil.copy(img_path, output_dir)

    logs.append(f"✅ Undersampling terminé pour {data_dir} → {output_dir} logs
```

Figure 37 : Code de under_sampling.ipynb

Main.ipynb

Script principal

Pipeline de traitement d'images avec undersampling

Ce script constitue le programme principal qui orchestre l'ensemble du processus de traitement des images. Il combine l'interface interactive de sélection de dossiers avec l'application de la technique d'undersampling, puis affiche un aperçu visuel des résultats.

Fonctionnement global :

Sélection interactive des dossiers sources :

Utilise la classe FolderSelector pour permettre à l'utilisateur de choisir les dossiers contenant les images à traiter

Interface graphique intégrée au notebook Jupyter

Application de l'undersampling :

Pour chaque dossier sélectionné, applique la fonction apply_undersampling

Réduit chaque dataset à un nombre cible d'images (target_count = 4999)

Copie les images sélectionnées dans un dossier de sortie commun

Validation visuelle :

Affiche un échantillon aléatoire d'images issues du traitementPermet de vérifier visuellement la qualité et la cohérence des images traitées

Gestion des chemins :

Calcule automatiquement le chemin racine du projet

Construit de manière portable le chemin de sortie (data/processed/Normal)

Assure la compatibilité multi-plateformes

Visualisation :

Affiche 5 images aléatoires du dataset final

Utilise matplotlib pour une visualisation rapide dans le notebook

Titres des images pour identification facile

Cas d'usage :

Préparation automatisée de datasets équilibrés

Pipeline de prétraitement pour l'apprentissage automatique

Interface utilisateur intuitive dans l'environnement Jupyter

```

● ● ●

import import_ipynb
import os
from select_folders import FolderSelector
from under_sampling import apply_undersampling
import time
import matplotlib.pyplot as plt
import random
from glob import glob

folder_selector = FolderSelector()
folder_selector.afficher()

while not folder_selector.fini:
    time.sleep(1)

# On récupère les dossiers sélectionnés
selected_dirs = folder_selector.chemins
print(f"Dossiers sélectionnés : {selected_dirs}")

target_count = 4999
# Récupère le chemin absolu du dossier courant du projet (là où se trouve ce notebook)
project_root = os.path.abspath(os.path.join(os.getcwd(), "..", ".."))

# Construit le chemin de sortie de façon portable
output_dir = os.path.join(project_root, "data", "processed", "Normal")
print("Chemin de sortie utilisé :", output_dir)

for data_dir in selected_dirs:
    print(f"▣ Traitement du dossier : {data_dir}")
    logs = apply_undersampling(data_dir, target_count, output_dir)
    for line in logs:
        print(line)

# Choisir le dossier à explorer
data_dir = selected_dirs[0]

images = glob(os.path.join(output_dir, '*.png'))
sample_images = random.sample(images, min(5, len(images)))

fig, axes = plt.subplots(1, len(sample_images), figsize=(15, 3))
if len(sample_images) == 1:
    axes = [axes]
for ax, img_path in zip(axes, sample_images):
    img = plt.imread(img_path)
    ax.imshow(img, cmap='gray')
    ax.axis('off')
    ax.set_title(os.path.basename(img_path))
plt.tight_layout()
plt.show()

```

Figure 38 : Code de Main.ipynb

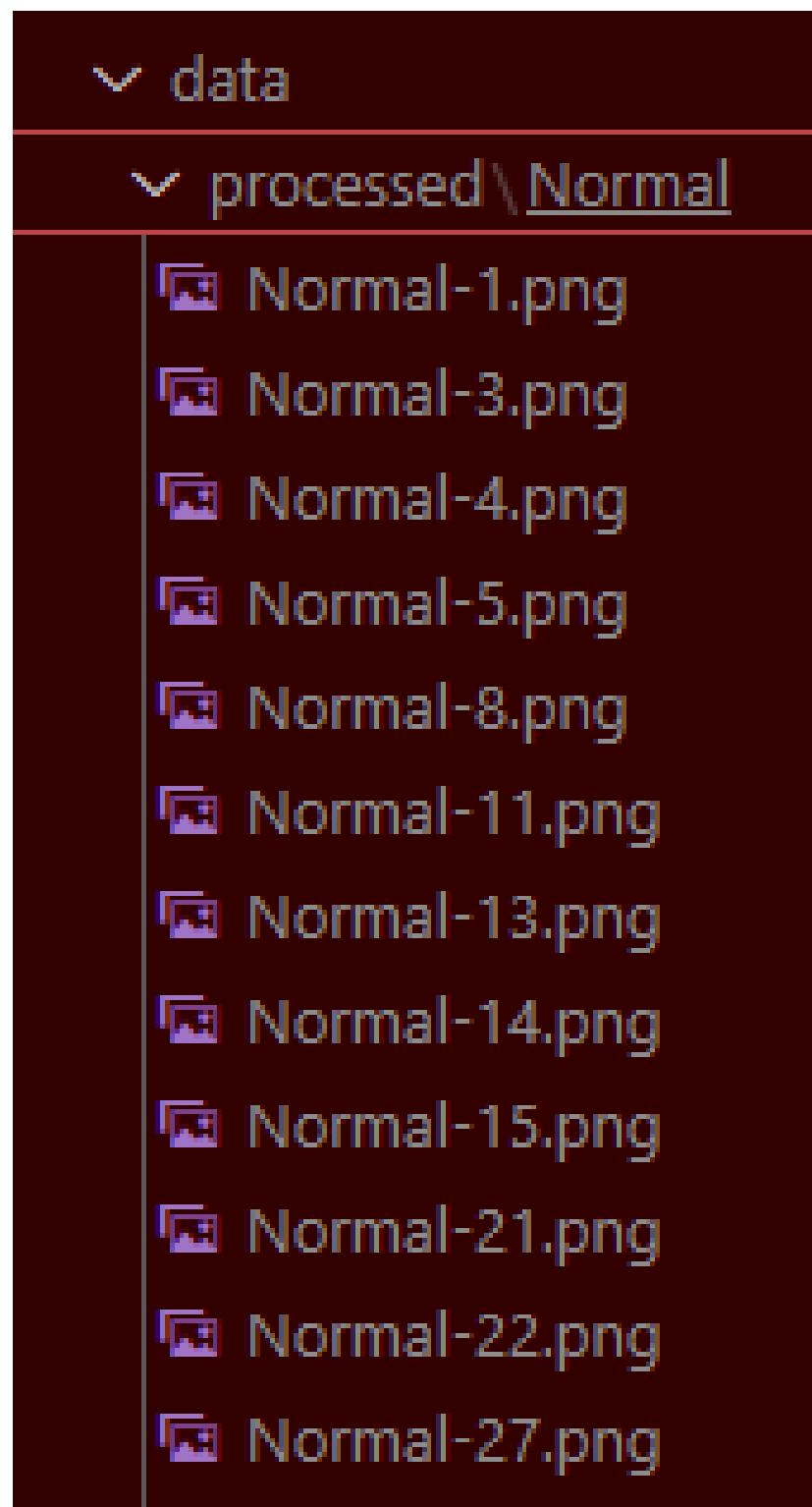


Figure 38 : Résultat de Main.ipynb

over_sampling.ipynb

Pour ce qui est de l'Oversampling, nous avons voulu tester directement avec Keras, et notamment la classe ImageDataGenerator, qui est faite pour cette occasion.

Dans le code suivant, nous avons utilisé des valeurs arbitraires, mais qui sont celles que l'on a pu voir dans la documentation du module.

L'utilisation de la méthode `flow_from_directory` apporte deux avantages majeurs :

- le chargement des images par batchs (ce qui permet alors directement d'anticiper la partie modélisation)
- la possibilité de redimensionner (`target_size`) et convertir les images (`color_mode`), d'un seul coup. C'est donc parfait pour nous initier à ce type de workflow.

```

import shutil
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator

base_dir = "C:/Users/rafae/Documents/GitHub/DS_COVID/data/raw/COVID-19_Radiography_Dataset/COVID-19_Radiography_Dataset/"
classes = ["COVID", "Normal", "Lung_Opacity", "Viral Pneumonia"]

# Crée un générateur d'augmentation
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2]
)

for cls in classes:    # Pour chaque classe

    images_dir = os.path.join(base_dir, cls, "images")
    parent_dir = os.path.dirname(images_dir)

    # On crée un dossier temporaire s'il n'existe pas
    tmp_class_dir = os.path.join(parent_dir, "tmp")

    os.makedirs(tmp_class_dir, exist_ok=True)

    # Copie quelques images dans le dossier temporaire (évite les doublons avec set)
    existing = set(os.listdir(tmp_class_dir))
    for img_name in os.listdir(images_dir)[:5]:
        if img_name not in existing:
            shutil.copy(os.path.join(images_dir, img_name), tmp_class_dir)

    # Pour flow_from_directory, il faut une structure /dossier/classe/images

    # Générateur batch
    generator = datagen.flow_from_directory(
        tmp_class_dir,
        target_size=(256, 256),
        color_mode='grayscale',
        batch_size=5,
        class_mode=None,
        shuffle=True
    )

    # Affiche un batch d'images augmentées
    batch = next(generator)
    plt.figure(figsize=(15, 3))
    for i in range(5):
        plt.subplot(1, 5, i+1)
        plt.imshow(batch[i].squeeze(), cmap='gray')
        plt.axis('off')
    plt.suptitle(f"Batch d'images augmentées (Keras) pour la classe {cls}")
    plt.show()

```

Figure 39 : Code de over_sampling.ipynb

Batch d'images augmentées (Keras) pour la classe Normal



Figure 40 : Résultat de over_sampling.ipynb

Etape 03 : Modélisation

Step 01 : Model Baseline

Un modèle baseline désigne un modèle simple ou rudimentaire utilisé en machine learning pour servir de point de comparaison aux modèles plus complexes développés par la suite. Son objectif principal n'est pas d'obtenir la meilleure précision possible, mais d'établir un niveau de performance minimal que tout modèle plus sophistiqué doit dépasser pour être jugé pertinent. (<https://www.lakera.ai/ml-glossary/baseline-models>)

- Un modèle baseline en classification peut consister à prédire systématiquement la classe majoritaire du jeu de données.
- En régression, il s'agira par exemple de prédire la valeur moyenne ou médiane de la variable cible, quelle que soit l'entrée.

Les modèles baseline ne cherchent pas la performance, mais servent à évaluer la difficulté intrinsèque du problème.

Exemples de modèles baseline :

- Pour classification : prédire la classe majoritaire, prédire au hasard, DummyClassifier.
- Pour régression : prédire la moyenne/médiane, DummyRegressor.

Nous avons choisi de tester les méthodes de machines learning suivantes : SVM, KNN et Forest Random.

SVM (Support Vector Machine): (<https://blent.ai/blog/a/modele-machine-learning-populaires>) (Figure 41)

Un SVM (Support Vector Machine) est un algorithme supervisé de classification et de régression.

Il cherche à tracer une frontière (hyperplan) qui sépare au mieux les classes dans un espace de caractéristiques, en maximisant la "marge" entre les points de chaque classe les plus proches de la frontière (les "vecteurs support").

C'est donc un modèle robuste, notamment pour les données de grande dimension, qui peut aussi intégrer des fonctions noyau ("kernel") pour gérer la non-linéarité : il projette les données dans un espace de plus grande dimension pour trouver une séparation optimale.

Utilisé en binaire ou multiclass, il est réputé pour sa précision sur des bases bien structurées.

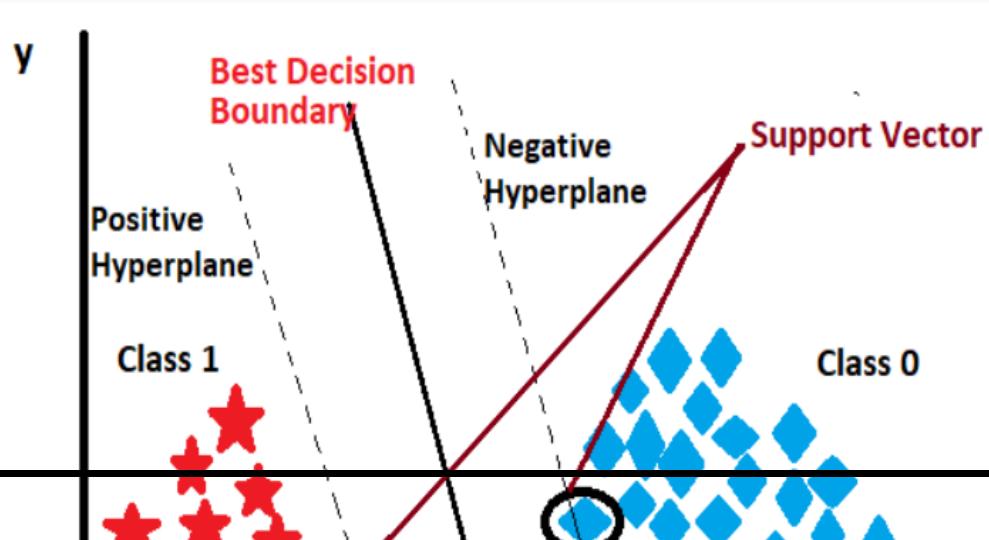


Figure 41: Principe de fonctionnement du SVM

K-Nearest Neighbors (KNN) :

Le KNN ("K plus proches voisins") est un algorithme non paramétrique supervisé.

Son principe : pour une nouvelle donnée à classer, il regarde les K points les plus proches dans le jeu d'entraînement (selon une mesure de distance, souvent euclidienne).

Il attribue alors la classe la plus fréquente parmi ces voisins ("majorité") en classification, ou la moyenne pour la régression. (**Figure 42**)

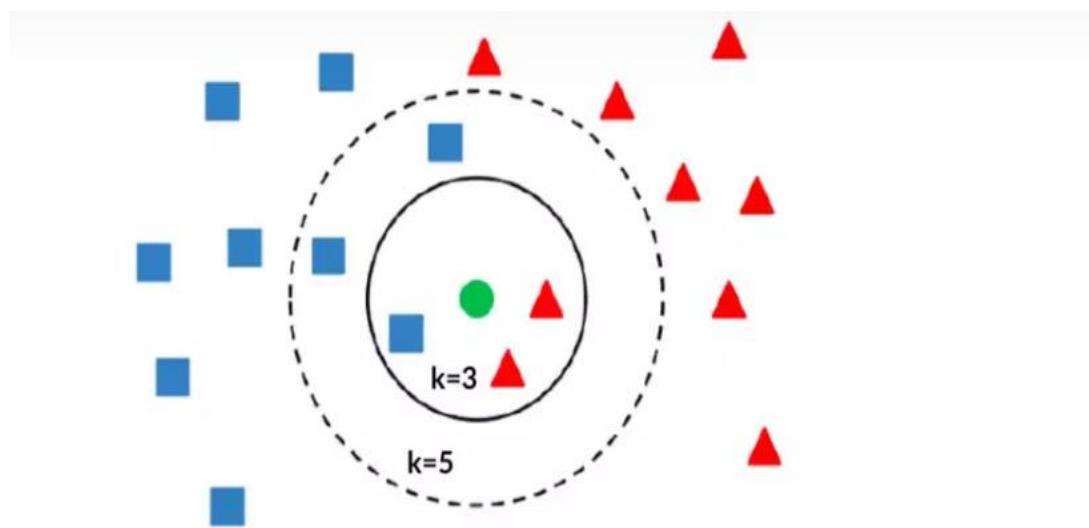


Figure 42 : Principe de fonctionnement des KNN

Random Forest (Forêt d'arbres Décisionnels) :

Le Random Forest est un algorithme "ensembliste" basé sur les arbres de décision.

Il construit une multitude d'arbres de décision sur des sous-échantillons aléatoires des données (méthode du "bagging").

Chaque arbre fait une prédiction, et dans le cas de la classification, la classe majoritaire parmi tous les arbres devient la prédiction finale ("vote majoritaire"). En régression, c'est la moyenne des prédictions.

Les arbres sont construits avec des sous-ensembles aléatoires de variables/features et de données, ce qui réduit le sur-apprentissage et augmente la robustesse.

Le modèle est très performant pour les données tabulaires et mixte (catégorielles / numériques), mais peut-être moins adapté à des données très éparses ou avec beaucoup de valeurs manquantes. (**Figure 43**)

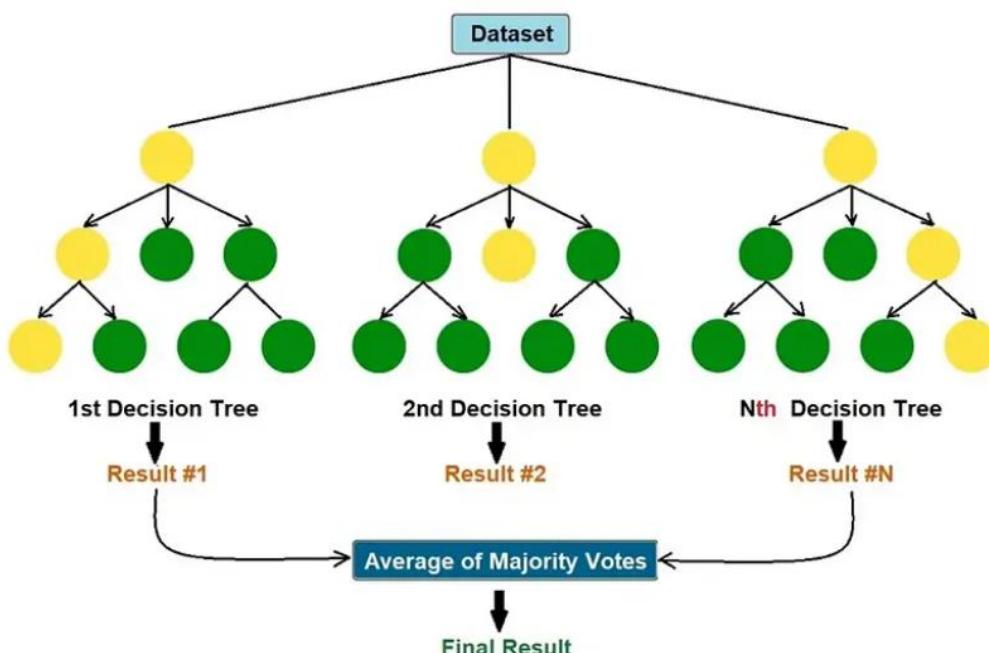


Figure 43 : Principe de fonctionnement Random Forest

Avantages et Inconvénients :

- **KNN** : très simple, mais peu efficace sur des volumes importants ou en présence de bruit.
- **SVM** : performant et robuste sur petites/moyennes dimensions, mais coûteux et difficile à régler.
- **Random Forest** : généraliste et précis, bien adapté aux données complexes, mais lourd en calcul et moins interprétable.

Résultats et Interprétation :