ZOIDBERG2.0

Léna Oudjman
Mozhgan Sigarchian
Faniry Ramavozatovo

0. Introdution

La pneumonie est une infection des poumons qui nécessite un diagnostic rapide et précis pour assurer un traitement efficace. La radiographie thoracique est l'un des outils les plus couramment utilisés pour le diagnostic de la pneumonie. Cependant, l'interprétation des radiographies peut être source d'erreurs humaines et nécessite une expertise médicale considérable. Dans ce contexte, les systèmes d'intelligence artificielle (IA) peuvent jouer un rôle crucial en aidant les cliniciens à diagnostiquer la pneumonie de manière plus rapide et plus précise.

1. Objectifs

Notre projet vise à développer et optimiser un modèle prédictif basé sur le Deep Learning capable de fournir un diagnostic précis et rapide sur la reconnaissance ou non de pneumonie bactérienne et virale sur des radiographies thoraciques.
Ce projet vise à aider les médecins, accélérer le diagnostic, améliorer les résultats cliniques et démontrer l'efficacité des méthodes d'apprentissage automatique.

2.Outils utilisés

On utilise des outils comme TensorFlow, PyTorch ou scikit-learn parce qu'ils offrent des algorithmes de machine learning et deep learning déjà optimisés, ce qui nous fait gagner du temps et évite les erreurs de programmation. Ils simplifient aussi le traitement des données et le calcul parallèle. On les utilise avec Python car c'est un langage simple et lisible, qui a une bonne docimentation et de nombreuses bibliothèques scientifiques puissantes.

Critères	TensorFlow	PyTorch	Scikit-Learn
Modèles Pré-entraînés	Oui (Keras)	Beaucoup de code open-source disponible	Oui
Documentation	Oui	Oui	Oui
Quantification	Oui	Simplifie l'entraînement	Modèles plus interprétables

Apprentissage	Complexe à maîtriser	Plus simple	Plus simple

Pour résumé :

TensorFlow est recommandé pour les projets nécessitant un déploiement en production robuste et bénéficiant de son écosystème complet et de ses outils de production.

PyTorch est préférable pour ceux qui recherchent la flexibilité, la facilité d'utilisation et un développement rapide, particulièrement dans un environnement de recherche ou académique.

Scikit-Learn peut être utilisé pour les étapes initiales de prétraitement des données et les analyses exploratoires avec des algorithmes de machine learning traditionnels, mais il ne suffira pas pour les modèles de deep learning nécessaires à cette tâche.

3.Organisation de notre repositorie

```
— ZOIDBERG2.0
  — src
    — bootstrap
    — utils
    — demo
    — result_tf
    — training_script
  — .gitignore
  — README.md
  — requirements.txt
```

4.Comment évaluer nos modéles ?

- **TP** (True Positives) : Nombre de vrais positifs
- **FN** (False Negatives) : Nombre de faux négatifs
- **FP** (False Positives) : Nombre de faux positifs
- **TN** (True Negatives) : Nombre de vrais négatifs

4.1 Loss (Fonction de perte)

La **loss** mesure l'erreur d'un modèle pendant l'entraînement. Il existe différents types de fonctions de perte, selon le type de problème.

Ici la loss sera : **Categorical Crossentropy** car c'est une fonction de perte utilisée pour les problèmes de classification multi-classes où chaque échantillon appartient à une seule

classe parmi plusieurs possibles. Cette fonction de perte est appropriée lorsquà les étiquettes de sortie sont encodées en one-hot.

La fonction de perte guide l'optimisation du modèle en ajustant les poids pour minimiser cette erreur.

4.2 Accuracy (Précision)

L'**accuracy** est le ratio des prédictions correctes sur le nombre total de prédictions.

C'est une mesure globale de performance, elle peut être trompeuse si les classes sont déséquilibrées. Pas assez efficace pour qu'on prenne que ca en compte.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

4.3 AUC (Area Under the Curve)

L'AUC fait référence à l'aire sous la courbe ROC (Receiver Operating Characteristic). La courbe ROC trace le taux de vrais positifs contre le taux de faux positifs à différents seuils de classification.

L'AUC fournit une mesure agrégée de la performance du modèle à tous les seuils de classification possibles. Une AUC de 1.0 indique une classification parfaite, tandis qu'une AUC de 0.5 indique une classification aléatoire.

4.4 Recall (Sensibilité ou Rappel)

Le **Recall** mesure la proportion de vrais positifs parmi les cas pertinents (vrais positifs et faux négatifs).Il indique la capacité du modèle à détecter les échantillons positifs.

$$\text{Recall} = \frac{TP}{TP+FN}$$

4.5 Precision (Précision)

La **Precision** mesure la proportion de vrais positifs parmi les cas prédits positifs (vrais positifs et faux positifs). Il indique la qualité des prédictions positives du modèle.

$$\text{Precision} = \frac{TP}{TP+FP}$$

4.6 Matrice de Confusion

	Prédit A	Prédit B	Prédit C
Réel A	TP_A	$\text{FP}_{A\to B}$	$\text{FP}_{A\to C}$
Réel B	$\text{FP}_{B\to A}$	TP_B	$\text{FP}_{B\to C}$
Réel C	$\text{FP}_{C\to A}$	$\text{FP}_{C\to B}$	TP_C

La **matrice de confusion** (ici pour 3 classes) est un tableau qui permet de visualiser les performances d'un modèle de classification. Elle compare les prédictions du modèle avec les valeurs réelles. Elle permet de calculer diverses métriques telles que la précision, le rappel, la spécificité, etc.

5.Manipulation de la Data

Les données fournies par Epitech étaient des données open source (vu sur Kaggle) nommées Chest_Xray, représentant des images (.jpeg) de scanners pulmonaires. Elles étaient réparties en deux classes : NORMAL, scanner de personnes non malades qui seront désignées par 0 par la suite, et PNEUMONIA, scanner de personnes atteintes de pneumonie désignées par 1.

La première chose à faire est de lancer `data.ipynb`. Cela permet de reformater nos données, de les trier, et d'avoir le même dataset de test grace a un seed. Cela est essentiel pour la comparaison de nos futurs modèles entraînés.

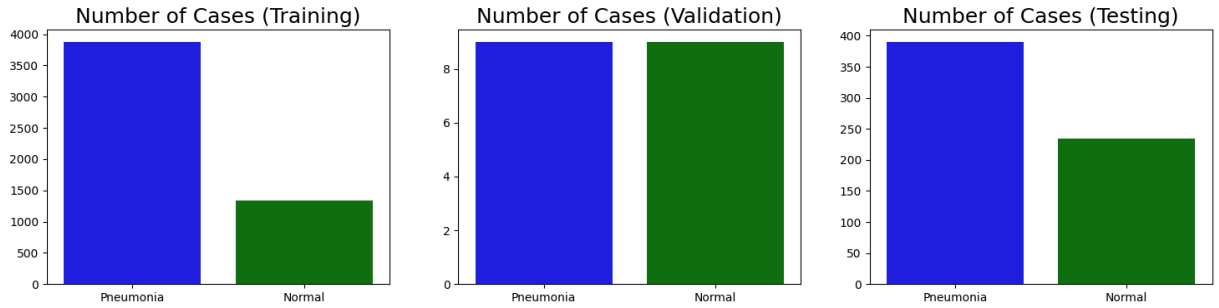

Figure 1 : Repartition de nos données de début

	Nb de data	Commentaires
Entraînement	5216	Il y a près de 3 fois plus de données de pneumonie que de données normales. Il y a deux types de scanners de pneumonie : virale et bactérienne.
Validation	18	Même nombre de cas de pneumonie et de cas normaux. De plus, il n'y a que des données de pneumonies bactériennes, pas de virales. Trés peu de données
Test	624	Répartition plutôt équilibrée.

Pour le dataset d'entraînement, les poids sont les suivants :
{0: 1.9448173005219984, 1: 0.6730322580645162}

- **0** : Non malade, sain. Ce poids est assez élevé, ce qui suggère que la classe "non malade" est sous-représentée dans le dataset par rapport à la classe "malade". Le poids élevé permet de donner plus d'importance aux échantillons de cette classe pendant l'entraînement, compensant ainsi leur faible fréquence.
- **1** : Malade dû à une bactérie. Ce poids est inférieur à 1, ce qui indique que la classe "malade dû à une bactérie" est sur-représentée dans le dataset. Le poids plus faible aide à réduire l'impact de cette classe dominante, équilibrant ainsi l'influence de chaque classe sur l'entraînement du modèle.

En raison de ces déséquilibres, nous avons décidé de séparer les données de pneumonie en deux catégories : virales et bactériennes.
De plus, on répartie les trois datasets (entraînement, validation et test) en respectant les proportions de 70%, 15% et 15% respectivement. Chaque dataset contient la même proportion de cas normaux, bactériens et viraux.

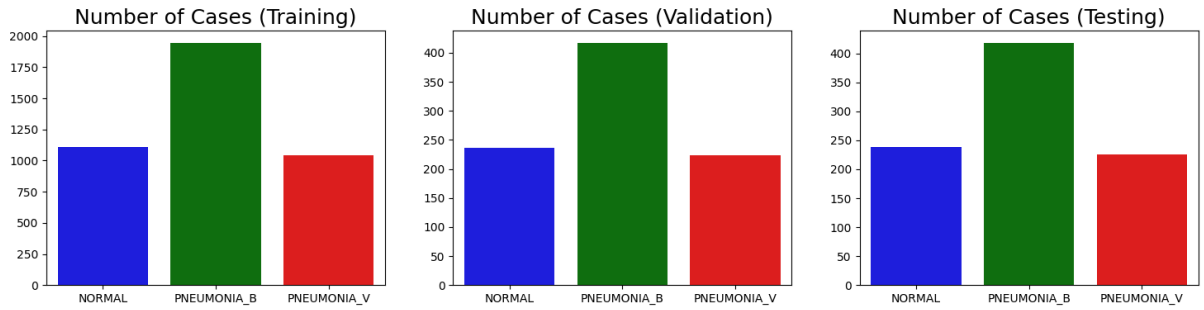

Figure 2 : Nos données avec les mêmes proportions

Nous pouvons voir que nos données d'entraînement sont plus equilibré :
{0: 1.2328519855595668, 1: 0.7023136246786632, 2: 1.307177033492823}

Pour le dataset d'entraînement, les poids sont les suivants :

- **0** : Non malade, sain. Ils ne sont pas majoritaires dans la représentation des données, donc leur poids est légèrement supérieur.
- **1** : Malade dû à une bactérie. Cette catégorie est la plus présente, donc son poids est plus faible.
- **2** : Malade dû à un virus. Ils ne sont pas majoritaires dans la représentation des données, donc leur poids est légèrement supérieur.

Attention, auparavant, j'avais plus de 1000 données d'entraînement en plus.

Pour rémedier à cette perte de données, nous allons faire de l'augmentation de données. nous allons prendre des données normal et viral et les flips pour obtenir des proportion egal de nos 3 classes.

Figure 3 : Exemple d’image Flip

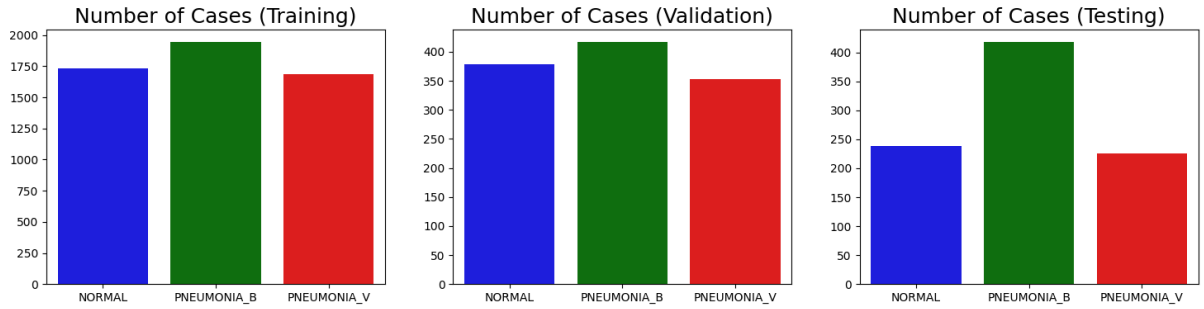

Figure 4 : Dernier version de données post augmentation

On obtient une répartition presque parfaite de nos données d'entraînement :
{0: 1.0315263360246059, 1: 0.9196229648671808, 2: 1.0602647698083383}

Pour le dataset d'entraînement, les poids sont les suivants :

- **0** : Non malade, sain. La valeur légèrement supérieure à 1 indique que les données de cette classe ne sont pas majoritaires dans le dataset, donc ce poids est légèrement plus élevé pour compenser leur sous-représentation.
- **1** : Malade dû à une bactérie. Étant donné que cette catégorie est la plus représentée dans le dataset, son poids est plus faible pour éviter un biais en faveur de cette classe pendant l'entraînement du modèle.
- **2** : Malade dû à un virus. Comme cette catégorie n'est pas majoritaire, le poids est légèrement plus élevé pour compenser leur sous-représentation, similaire à la classe "non malade".

6.Tensorflow model (Léna)

Pour commencer, on a importé les chemins nécessaires pour accéder aux données et configurations, puis on configuré TensorFlow pour tirer parti du GPU, si disponible, afin d'accélérer l'entraînement.

La partie centrale de notre code est la classe `CustomModelTrainer`, qu'on a crée pour faciliter l'entraînement du modèle. Cette classe me permet de spécifier les chemins pour les données d'entraînement, de validation et de test, ainsi que divers paramètres comme la taille des images (150,150), le nombre d'époques (on a choisi 50 pour commencer, sachant qu'on a impleté un EarlyStop), le batch (32 pour train et validation, le nombre d'image pour le test). Les données sont preparées en utilisant `ImageDataGenerator`, qui fait de l'augmentation pour enrichir les données et améliorer la performance du modèle.

Pour le modèle lui-même, on a opté pour un réseau de neurones convolutionnel (CNN). On utilise des couches de convolution pour extraire les caractéristiques des images, suivies de couches de normalisation et de pooling pour affiner les données et éviter le surapprentissage. Ensuite, j'ai ajouté des couches denses pour effectuer les prédictions finales avec une sortie de 3 unités (pour nos 3 classes: Normal, Bactérienne, Viral) et une fonction d'activation "softmax" pour transformer nos sorties en probabilité entre 0 et 1.

On a inclus plusieurs callbacks pour rendre l'entraînement plus efficace. `ModelCheckpoint` sauvegarde le modèle à la fin de chaque époque, `ReduceLROnPlateau` ajuste le taux d'apprentissage si le modèle ne progresse plus, `EarlyStopping` arrete l'entrainement si la valeur de la loss ne progresse plus (pour le model presenté, on a retirer le EarlyStopping car au bout de 5 epochs le model arretait de s'entrainer et le resultat était vraiment pas bon) et `TensorBoard` enregistre les détails de l'entraînement pour suivre la performance.

Une fois l'entraînement terminé, on sauvegarde le modèle pour pouvoir l'utiliser plus tard.

En résumé, on a conçu ce modèle pour préparer les données, entraîner un CNN, et assurer un suivi efficace durant l'entraînement en sauvegardant le modèle et en ajustant les paramètres au besoin.

Après avoir entraîné le modèle, nous nous assurons qu'il n'y a pas de différence significative entre les valeurs des métriques d'entraînement et de validation.
Nous pouvons enfin appliquer le modèle à nos données test. On obtient une matrice de confusion, on calcule ensuite la Precison , la Recall et le F1-score.

	Predict Sain	Predict Bacterien	Predict Viral
True Sain	0	238	0
True Bacterien	0	418	0
True Viral	0	225	0

Matrice de confusion avec Early Stop

	Predict Sain	Predict Bacterien	Predict Viral
True Sain	230	2	6
True Bacterien	27	344	47
True Viral	26	76	123

Matrice de confusion Finale

F1-score: [0.88291747 0.81904762 0.61346633]
Precision: [0.81272085 0.81516588 0.69886364]
Recall: [0.96638655 0.82296651 0.54666667]

Ces résultats indiquent que le modèle a une très bonne performance pour les images normales, avec un F1-score élevé de 0.8829, ce qui suggère une bonne balance entre la précision et le rappel. Les images bacteriennes ont également une performance solide, avec un F1-score de 0.8190, montrant une légère amélioration par rapport à la Classe 1 (normal) en termes de rappel. Cependant, pour la Classe 3, les métriques sont moins favorables, avec un F1-score de 0.6135, une précision de 0.6989 et un rappel de 0.5467. Cela indique que le modèle a plus de difficultés à prédire correctement les pneumonies virales, ce qui pourrait être dû à un déséquilibre des données ou à des caractéristiques moins distinctives pour cette classe.

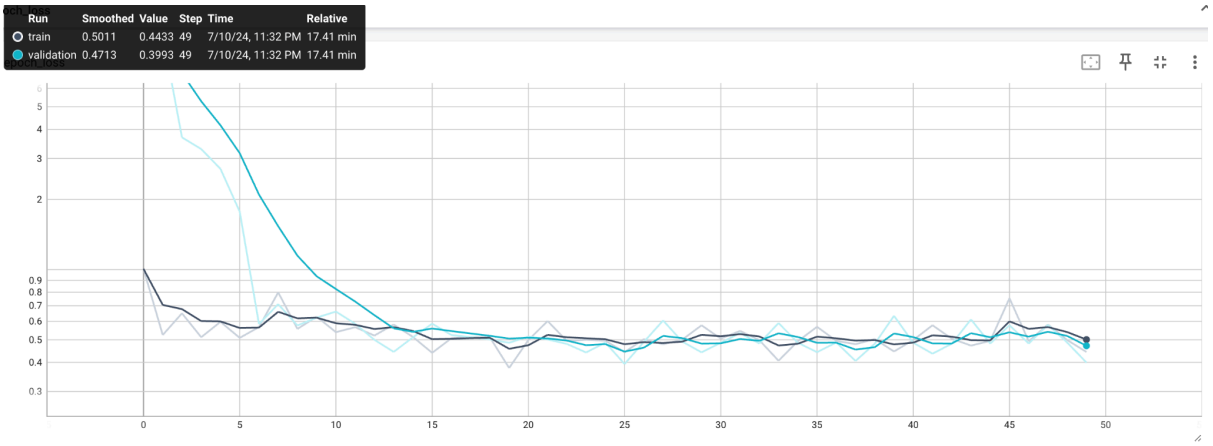

Figure 5 : Graphique de la loss par epoch

On voit que la loss diminue au fur et à mesure que les epochs avancent : notre model apprend bien.

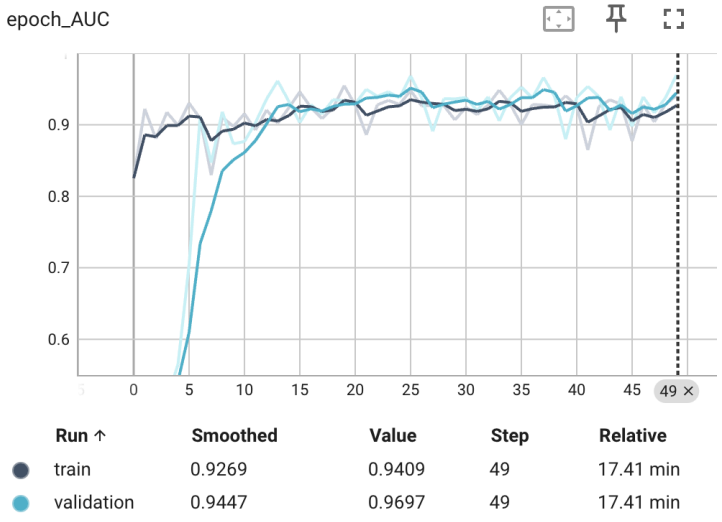

Figure 6 : Graphique AUC lors de l'entrainement en fonction de l'epoch

Les valeurs se rapprochent de 1 plus les epochs passent, le model est donc encourageant.

Les résultats de l'entraînement du modèle montrent une performance prometteuse pour la classification des images. Après 50 époques d'entraînement, le modèle a atteint une précision de l'ordre de 81% tant sur l'ensemble d'entraînement que sur l'ensemble de validation, ce qui indique une bonne capacité de généralisation. La loss moyenne diminue au fil des époques, ce qui montre que le modèle apprend efficacement les caractéristiques des images. Les métriques supplémentaires, telles que le F1-score de 0.819, la précision de 0.815 et le rappel de 0.823, soulignent également un bon équilibre entre la capacité du modèle à identifier correctement les classes positives et négatives, minimisant ainsi les faux positifs et faux négatifs. Cela indique que le modèle est assez bien calibré pour cette tâche de classification spécifique.

Modèle	Précision	Recall	F1-score	Accuracy	AUC	loss
TF_cnn	**[0.81272085 0.81516588 0.69886364]**	**[0.96638655 0.82296651 0.54666667]**	**[0.88291747 0.81904762 0.61346633]**	**0.7834**	**0.9351**	**0.4870**

7.Pytorch model (Mozgan)

7.1 Objectif d'Utiliser un SimpleCNN

Un SimpleCNN est une version simplifiée des CNN, idéale pour le traitement rapide et efficace des images. Avec moins de paramètres et une architecture plus légère. Par rapport aux réseaux entièrement connectés et aux arbres de décision, le SimpleCNN nécessite moins de prétraitement et est plus adapté aux images. Les RNN, conçus pour les données séquentielles, ne conviennent pas aux tâches d'imagerie. Le SimpleCNN est préféré pour sa simplicité et son efficacité.

7.2 Nombre d'Époques

Le nombre d'époques indique combien de fois le modèle passe par toutes les données d'entraînement. Nous avons choisi 10 époques par défaut. Plus d'époques peuvent améliorer le modèle, mais nécessitent plus de temps.

7.3 Taille des Batchs

La taille des batchs est de 32 dans notre modèle. Cela affecte :

- **Stabilité de l'entraînement** : Des lots plus grands offrent des mises à jour plus stables.
- **Mémoire GPU** : Des lots plus grands utilisent plus de mémoire.
- **Vitesse** : Des lots plus grands peuvent accélérer l'entraînement.

7.4 Évaluation de la Performance du Modèle

	True Sain	True Bacterien	True Viral
Predict Sain	144	79	15
Predict Bacterien	0	415	3
Predict Viral	2	209	14

Matrice de confusion Finale

Nous avons utilisé plusieurs mesures pour évaluer le modèle :

F1-Score

- **Valeurs obtenues** : 0.75, 0.74, 0.11

Les scores de 0.75 et 0.74 sont bons, mais le score de 0.11 indique que le modèle ne performe pas bien pour la troisième classe.

Précision

- **Valeurs obtenues** : 0.61, 0.99, 0.06

La précision montre combien de fois le modèle a raison parmi ses prédictions. Une précision élevée pour la deuxième classe est bonne, mais la très faible précision pour la troisième classe signifie beaucoup d'erreurs.

Rappel

- **Valeurs obtenues** : 0.99, 0.59, 0.44

Le rappel mesure combien de fois le modèle détecte correctement les éléments d'une classe. Un rappel élevé pour la première classe est excellent, mais la valeur plus basse pour la troisième classe montre que le modèle rate beaucoup de cas de cette classe.

7.5 Conclusion et Améliorations

- **Résultats généraux** : Le modèle fonctionne bien pour les deux premières classes mais a des difficultés avec la troisième. Cela peut être dû à un déséquilibre entre les classes ou à des caractéristiques difficiles à apprendre pour la troisième classe.
- **Améliorations possibles** : Pour améliorer les performances, nous pourrions équilibrer les classes dans les données, ajuster les paramètres d'entraînement ou modifier l'architecture du modèle.

7.6 Result

Modèle	Précision	Recall	F1-score	Accuracy	AUC	loss
TF_cnn	**[0.81272085 0.81516588 0.69886364]**	**[0.96638655 0.82296651 0.54666667]**	**[0.88291747 0.81904762 0.61346633]**	**0.7834**	**0.9351**	**0.4870**

8. Scikit-Learn model (Faniry)

La première chose qu'on a fait c'est de charger et prétraiter les données en les redimensionnant et les stocker dans une liste avec leurs labels correspondants afin de faciliter l'entraînement.

Pour le prétraitement et la réduction de dimension des données d'images, on a utilisé "StandardScaler" pour une distribution de pixel centrée avec une moyenne de 0 et une variance de 1 et l'ACP (Analyse en Composantes Principales) qui réduit la dimension des données à 200 composantes principales. Ces étapes réduisent considérablement la complexité des données, ce qui peut accélérer l'entraînement des modèles.

Après avoir chargé et prétraité nos données d'images, il est essentiel de comprendre les caractéristiques de ces images, notamment leurs dimensions. Nous avons visualisé et analysé la distribution des dimensions des images en utilisant un histogramme 2D pour s'assurer que les nouvelles dimensions des images sont appropriées.

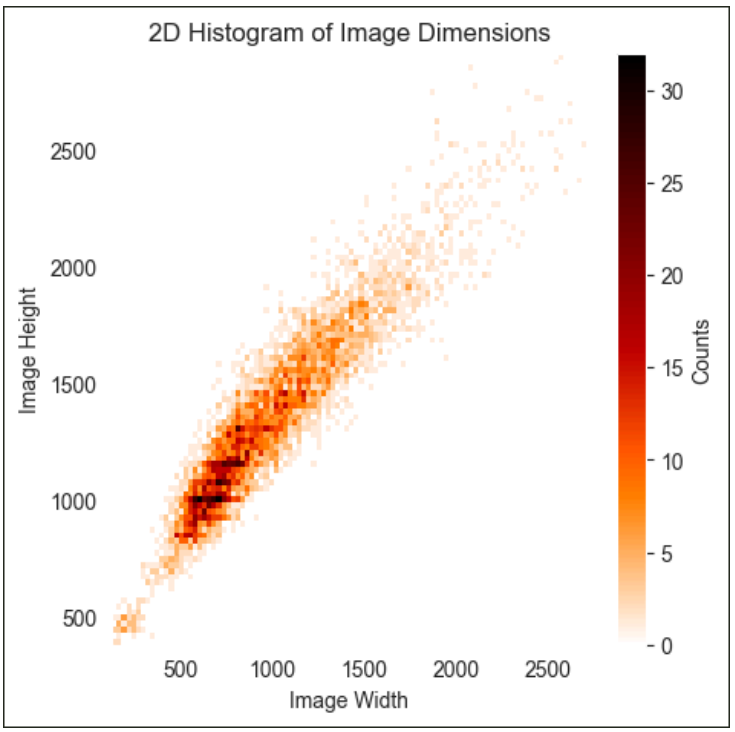

Figure 7: Histogramme 2D

Le résultat de l'histogramme 2D montre que la majorité des images ont des dimensions similaires, indiquant que nos images ont toutes été bien redimensionnées afin d'améliorer la performance du modèle.

Pour le modèle scikit-learn, nous avons choisi d'utiliser "SVM" avec un noyau RBF (Radial Basis Function). Nous avons utilisé une recherche randomisée avec "RandomizeSearchCV" pour optimiser les hyperparamètres C et gamma en définissant une grille de paramètres. Dans notre cas, la grille de paramètres que nous avons définie est:

{'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001]}

Cette approche nous permet de trouver les meilleures combinaisons de ces hyperparamètres et de maximiser la précision du modèle. Ensuite, la recherche a été configurée pour effectuer 10 itérations afin d'explorer différentes combinaisons de `C` et `gamma`.

En utilisant les meilleurs hyperparamètres trouvés, nous avons entraîné le modèle final sur les données d'entraînement puis les performances du modèles ont été évaluées sur les ensembles de test et de validation en générant ces rapports de classification. Pour plus de précision, le F1 score a été calculé pour l'ensemble de tests afin de fournir une mesure de performance globale.

Après avoir entraîner et évaluer le modèle, on obtient un résultat assez convaincant de nos données:

```
...   Test Classification Report:
                    precision    recall  f1-score   support

            NORMAL       0.91      0.93      0.92       238
       PNEUMONIA_B       0.77      0.88      0.82       418
       PNEUMONIA_V       0.64      0.45      0.53       225

          accuracy                           0.78       881
         macro avg       0.77      0.75      0.76       881
      weighted avg       0.77      0.78      0.77       881

      F1 Score: 0.77
```

Figure 8: test classification du modèle SVM

L'exactitude globale du modèle est de 0.75, soit 75% des prédictions globales du modèle sont correctes.
Suite au calcul de Précision, de Recall et le F1-score, nous avons tracé la matrice de confusion pour les données de test afin de visualiser les erreurs de classification.

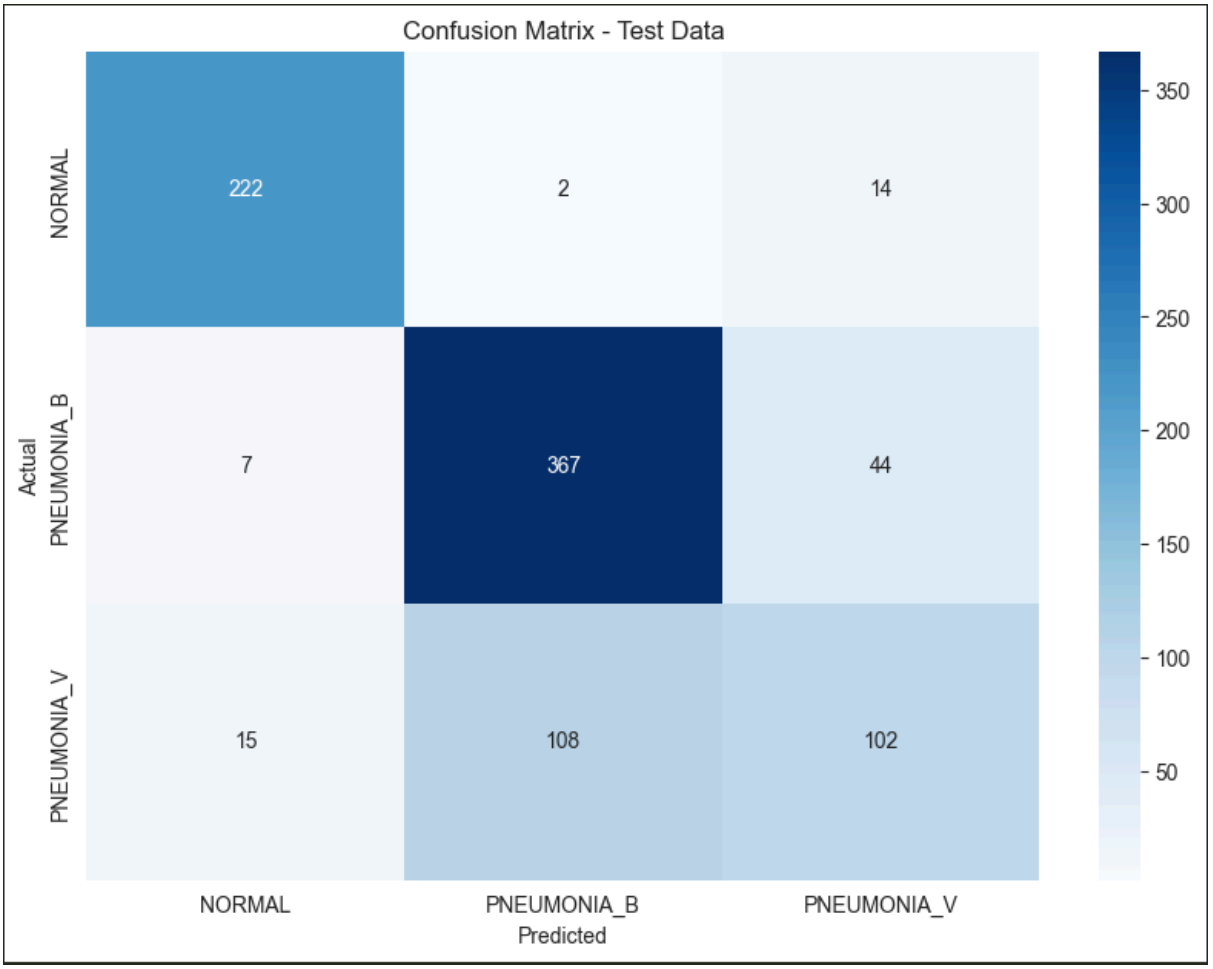

Figure 9: Matrice de confusion pour le modèle SVM

En plus du modèle **SVM**, nous avons exploré deux autres algorithmes de classification : le modèle **Random Forest** qui est un ensemble d'arbres de décision utilisés pour la classification et la régression, et le modèle de **Régression Logistique** qui est un modèle de classification linéaire qui est souvent utilisé pour des problèmes de classification binaire, mais il peut également être adapté pour des problèmes de classification multiclasse.
Voici une représentation des résultats obtenu après l'entraînement des modèles:

```
Test Classification Report:
              precision    recall  f1-score   support

      NORMAL       0.82      0.80      0.81       237
   PNEUMONIA_B       0.70      0.88      0.78       417
   PNEUMONIA_V       0.67      0.35      0.46       223

    accuracy                           0.73       877
   macro avg       0.73      0.68      0.68       877
weighted avg       0.72      0.73      0.71       877

F1 Score: 0.71
```

Figure 10: Test classification du modèle RandomForestClassifier

```
Test Classification Report:
              precision    recall  f1-score   support

      NORMAL       0.85      0.83      0.84       237
   PNEUMONIA_B       0.70      0.91      0.79       417
   PNEUMONIA_V       0.72      0.35      0.47       223

    accuracy                           0.74       877
   macro avg       0.76      0.70      0.70       877
weighted avg       0.75      0.74      0.72       877

F1 Score: 0.72
```

Figure 11: Test classification du modèle Logistic Regression

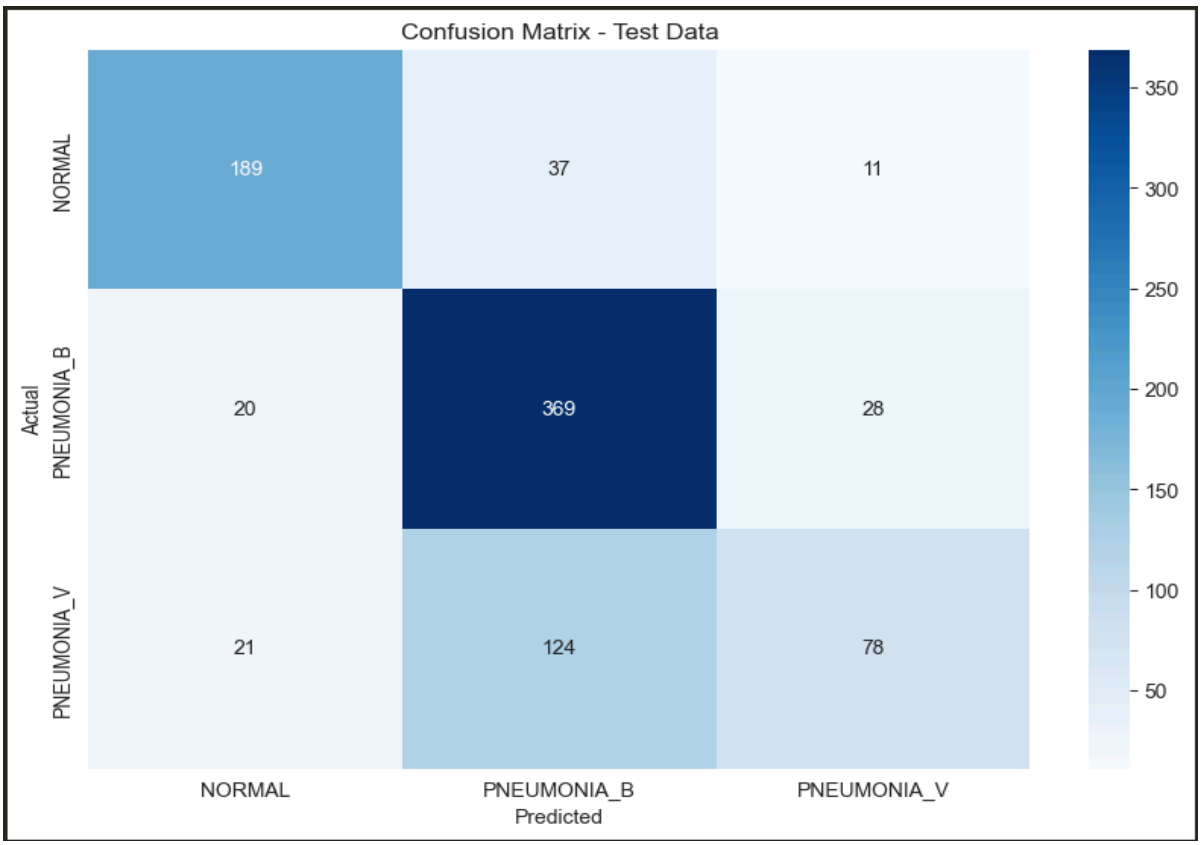

Figure 12: Matrice de confusion pour le modèle Random Forest Classifier

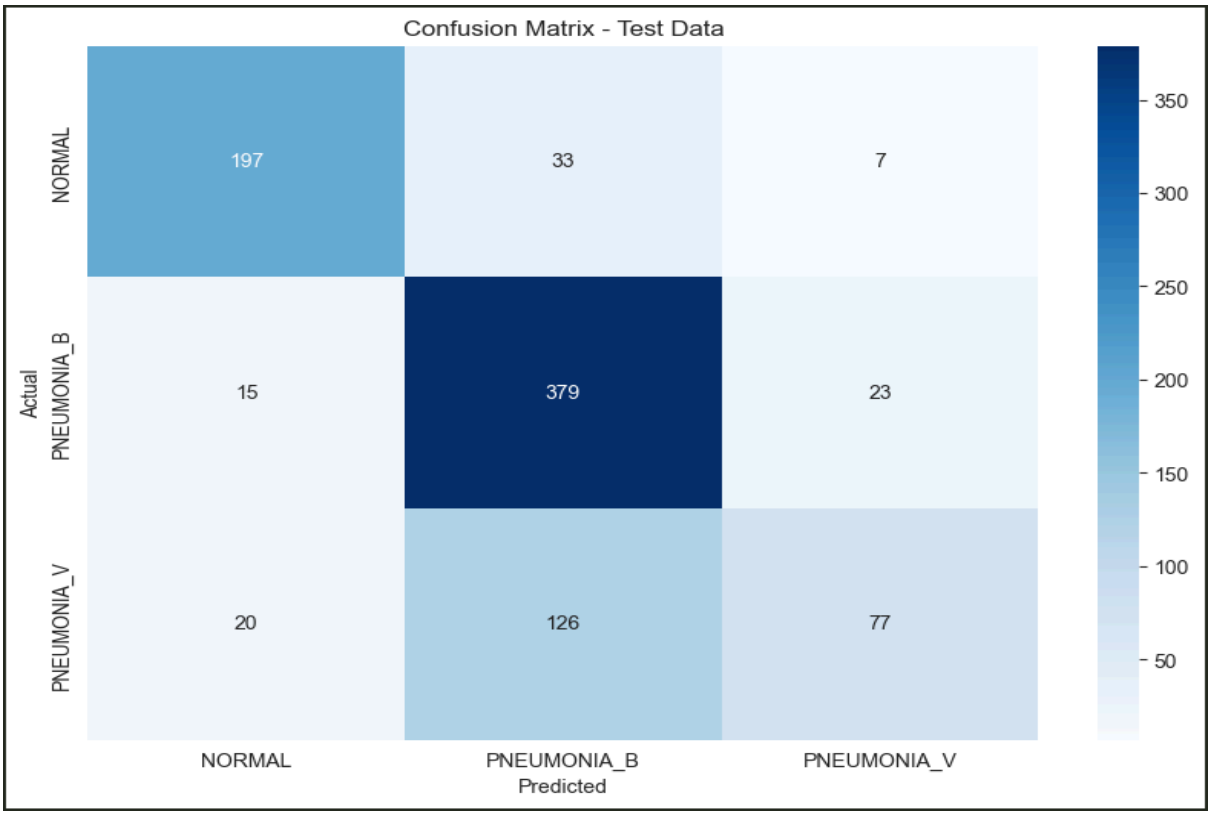

Figure 13: Matrice de confusion pour le modèle Logistic Regression

8.1 Comparaison des modèles scikit-learn

Afin de déterminer lequel de ces modèles performe le mieux sur notre jeu de données, nous avons comparé leurs résultats en termes de précision, rappel, F1-score et précision globale. Voici un tableau récapitulatif des performances de chaque modèle :

Modèle	Précision	Recall	F1-score	Accuracy
SVM	**0.77**	**0.78**	**0.77**	**0.78**
Random Forest	**0.72**	**0.73**	**0.71**	**0.73**
Logistic Regression	**0.75**	**0.74**	**0.72**	**0.74**

Le modèle **SVM** est le meilleur parmi les trois. Il présente la meilleure exactitude globale (0.78), le meilleur F1-score (0.77), et il équilibre bien la précision (0.77) et le rappel (0.78).

9.Result

Sur les listes, la premieres valeurs correspond au Sain, la deuxieme au Bacterien et la troisieme au viral.

Modèle	Précision	Recall	F1-score	Accuracy	AUC
TF_cnn	**[0.81272085 0.81516588 0.69886364]**	**[0.96638655 0.82296651 0.54666667]**	**[0.88291747 0.81904762 0.61346633]**	**0.7834**	**0.9351**
SVM	**0.77**	**0.78**	**0.77**	**0.78**	
Random Forest	**0.72**	**0.73**	**0.71**	**0.73**	
Logistic Regression	**0.75**	**0.74**	**0.72**	**0.74**	
Torch_cnn	**[0.6050, 0.9928, 0.0622]**	**[0.9863, 0.5903, 0.4375]**	**[0.7500, 0.7404, 0.1089]**	**0.650**	

10.Styles du repositorie

Lors de la gestion et du développement de notre code, il est essentiel de suivre des pratiques de codage strictes pour garantir la cohérence, la lisibilité et la maintenabilité. À

cette fin, nous utilisons des librairies spécifiques pour le style et la qualité du code, telles que **Black** et **Ruff**.
Black est un formateur de code automatique qui applique des conventions de style PEP 8, assurant ainsi un code uniformément formaté sans nécessiter de discussions sur le style.
Ruff est un linter qui détecte les erreurs et les incohérences potentielles dans le code, en aidant à maintenir un code propre et sans bugs. L'utilisation de ces outils permet de standardiser notre codebase, facilitant la collaboration et la révision du code au sein de l'équipe.

11.Documentation

https://pytorch.org/docs/stable/index.html
https://www.tensorflow.org/tutorials/keras/classification?hl=fr
https://datascientest.com/matrice-de-confusion
https://github.com/Lenoush/tuto_usefull/blob/main/README_pyenv.md