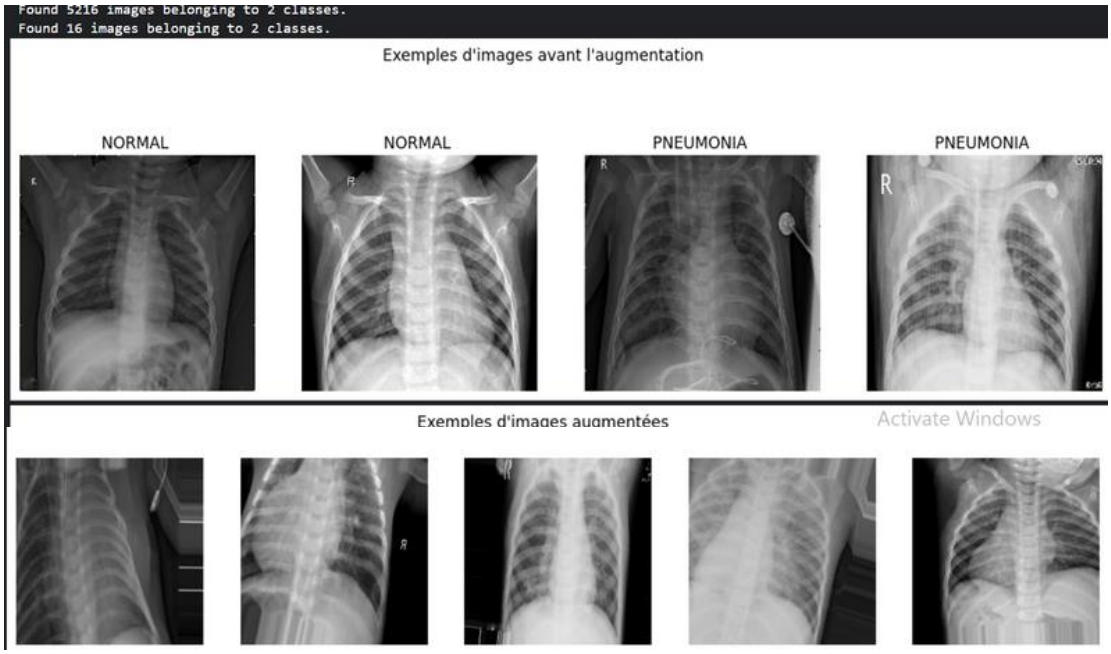
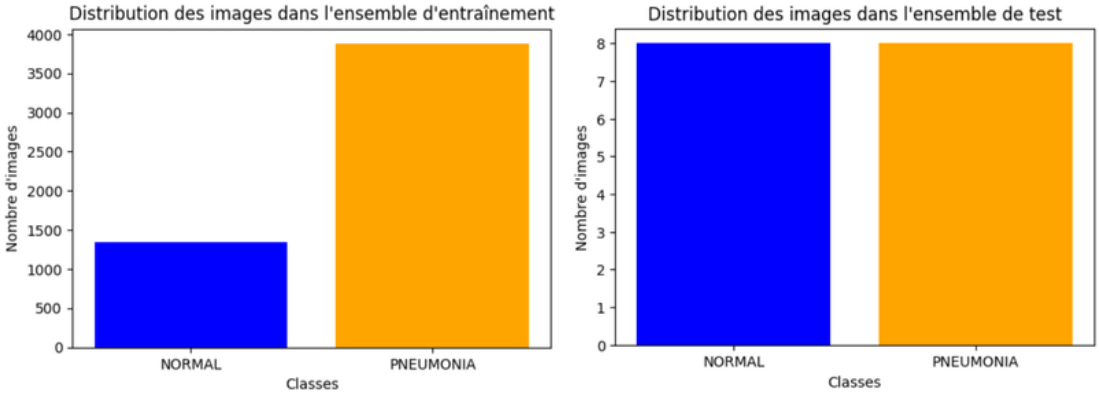
RAPPORT DE PROJET

DEEP LEARNING

01. Importation prétraitement et visualisation de données



02. Visualisation de la répartition des données



On a prétraité les données en les normalisant et en les augmentant pour enrichir le jeu d'entraînement. On a visualisé des exemples d'images avant et après l'augmentation pour valider leur pertinence. La distribution montre un déséquilibre dans l'ensemble d'entraînement (plus de cas de "PNEUMONIA"), ce qui pourrait biaiser les résultats. Les callbacks utilisés renforcent l'efficacité et la stabilité du modèle

03. Modele DenseNet121

On a utilisé DenseNet121 avec un modèle de base pré-entraîné pour détecter les cas de pneumonie.
Les résultats finaux montrent une précision de 88,78 % sur l'ensemble de test, avec une perte de 0,3912. La matrice de confusion révèle une meilleure classification des cas de pneumonie que des cas normaux.

1. Prétraitement des Données

Pour améliorer la robustesse et réduire le surapprentissage, les données ont été prétraitées en appliquant des techniques d'augmentation :

- Train Dataset :
  - Redimensionnement des images à 224x224 pixels.
  - Normalisation des pixels (rescale 1/255).
  - Augmentation des données : rotation, décalage, cisaillement, zoom, et inversion horizontale.
- Validation et Test Dataset :
  - Redimensionnement des images à 224x224 pixels.
  - Normalisation des pixels sans augmentation.

Les ensembles d'entraînement, de validation et de test proviennent d'un dataset public organisé dans des répertoires correspondant aux classes "Normal" et "Pneumonia".

2. Modèle DenseNet121

Le modèle DenseNet121 a été utilisé comme base grâce à ses couches convolutives puissantes pour l'extraction de caractéristiques.

1.Architecture de Base :

- DenseNet121, pré-entraîné sur ImageNet, a été intégré sans les couches fully connected.
- Les poids du modèle DenseNet121 ont été gelés dans un premier temps pour éviter de perturber les connaissances acquises.

2.Couches Supérieures Ajoutées :

- Une couche de GlobalAveragePooling2D pour réduire les dimensions tout en conservant l'information.

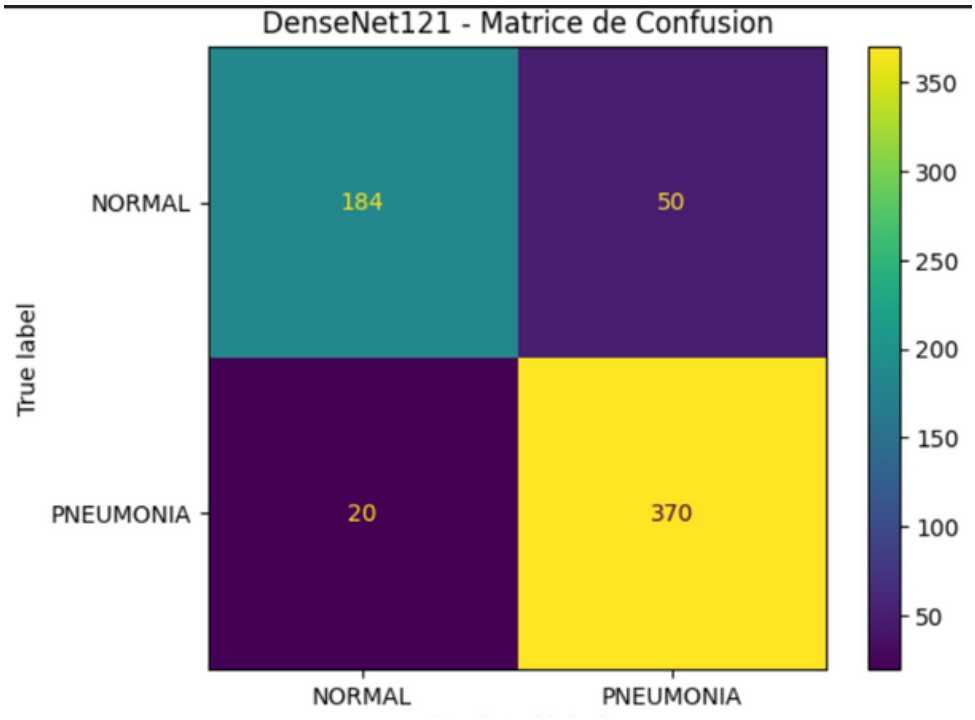
- Une couche dense avec 128 neurones, une activation ReLU et une régularisation L2.
- Une couche de dropout (60%) pour réduire le surapprentissage.
- Une couche finale avec 2 neurones et une activation softmax pour la classification.

- **Optimisation et Compilation :**
  - Fonction de perte : categorical_crossentropy pour les données catégoriques à deux classes.
  - Optimiseur : Adam avec un taux d'apprentissage initial de 1e−41e-41e−4.
  - Métrique : accuracy.

3. Stratégie d'Entraînement

L'entraînement s'est déroulé en deux étapes :

- Entraînement Initial :
  - Les couches de DenseNet121 sont gelées, seules les couches ajoutées sont entraînées.
  - 30 époques avec les callbacks suivants :
    - EarlyStopping : Arrête l'entraînement si la perte de validation ne diminue pas après 5 époques.
    - ReduceLROnPlateau : Réduit le taux d'apprentissage si la perte de validation ne s'améliore pas après 3 époques.
- **Fine-Tuning :**
  - Certaines couches de DenseNet121 (supérieures à la moitié des couches) ont été dégelées.
  - Réduction du taux d'apprentissage à 1e−51e-51e−5.
  - 10 époques supplémentaires pour ajuster les poids pré-entraînés.



03.

Le modèle **DenseNet121** semble très performant avec une précision de **88,78%**.
La perte de **0,3912** confirme que, bien que le modèle fasse quelques erreurs, ces erreurs sont relativement faibles.
Le modèle est donc bien adapté à la tâche de classification des radiographies, avec une bonne capacité de généralisation.

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 215ms/step - accuracy: 0.8437 - loss: 0.5093
DenseNet121 - Test Accuracy: 88.78%
DenseNet121 - Test Loss: 0.3912
```

04.

Modéle MobileNetV2

on a utilisé MobileNetV2 comme base pré-entraînée pour extraire des caractéristiques importantes , Voici les étapes dans ce modèle :

1.Préparation des données avec augmentation d'images:

- ImageDataGenerator génère des lots d'images avec des transformations pour augmenter la diversité du jeu de données et éviter le surapprentissage (ex : rotations, décalages, inversions horizontales, etc.).
- Les images sont redimensionnées à 224x224 pixels et mises à l'échelle entre 0 et 1 pour les entrées du réseau.

2. Création du modèle:

- **MobileNetV2** : Ce modèle pré-entraîné est utilisé sans la couche de classification supérieure (enlevant la partie top). Cela permet de réutiliser les poids appris sur ImageNet pour des tâches similaires.
- Les couches du modèle de base sont gelées pour empêcher leur réentraînement, et seule la partie supérieure du modèle est formée.
- Le modèle se compose de :

Flatten : Aplatisse les sorties du modèle MobileNetV2 en un vecteur.
Dense : Une couche entièrement connectée avec 128 neurones et une régularisation L2 pour éviter l'overfitting.
Dropout : Permet de ne pas utiliser certaines unités pendant l'entraînement (probabilité de 60%) pour réduire le surapprentissage.
Dense : La couche de sortie avec 2 neurones, activée par softmax pour la classification.

. **3.Compilation du modèle:**

- Le modèle est compilé avec l'optimiseur Adam, la fonction de perte categorical_crossentropy (adaptée aux tâches de classification multi-classes), et la métrique accuracy.

4. Callbacks:

- ReduceLROnPlateau : Réduit le taux d'apprentissage lorsque la validation de la perte ne s'améliore plus.
- EarlyStopping : Arrête l'entraînement si la perte de validation ne s'améliore pas pendant un certain nombre d'époques (patience de 5 époques), et restaure les meilleurs poids du modèle.

5. Entraînement du modèle:

- Le modèle est formé pendant 30 époques, avec les données d'entraînement et de validation, et les callbacks sont utilisés pour optimiser l'entraînement.

6. Fine-tuning:

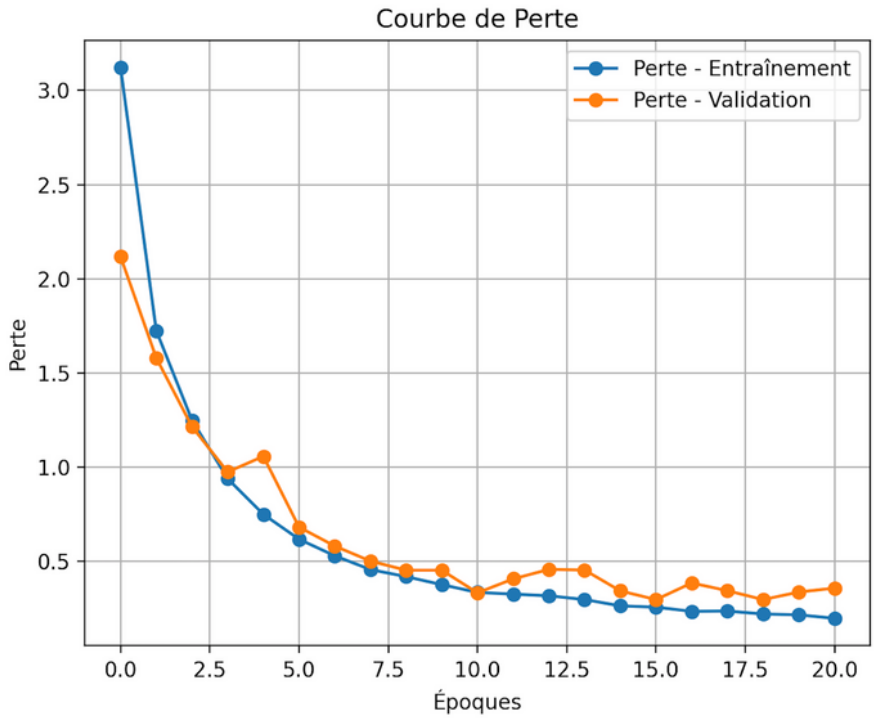
- Après l'entraînement initial, certaines couches du modèle pré-entraîné sont déverrouillées pour permettre un ajustement plus précis du modèle (fine-tuning).
- Seules les couches situées dans la moitié supérieure du modèle pré-entraîné sont formées, en réutilisant un taux d'apprentissage plus faible (1e-5).

04.

```
Run  modele2_version2
/Users/mac/anaconda3/envs/imenEnvironnementTP1/bin/python /Users/mac/PycharmProjects/pythonProject/modele2_version2.py
Found 5216 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
2025-01-01 19:34:06.174420: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2025-01-01 19:34:08.616854: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/30
163/163 [==============================] - 168s 1s/step - loss: 3.0357 - accuracy: 0.8894 - val_loss: 2.2179 - val_accuracy: 0.7500
Epoch 2/30
163/163 [==============================] - 162s 993ms/step - loss: 1.6975 - accuracy: 0.9281 - val_loss: 1.5788 - val_accuracy: 0.8125
Epoch 3/30
163/163 [==============================] - 146s 894ms/step - loss: 1.2319 - accuracy: 0.9281 - val_loss: 1.2849 - val_accuracy: 0.7500
Epoch 4/30
163/163 [==============================] - 147s 899ms/step - loss: 0.9349 - accuracy: 0.9323 - val_loss: 0.9307 - val_accuracy: 0.8750
Epoch 5/30
163/163 [==============================] - 146s 894ms/step - loss: 0.7466 - accuracy: 0.9360 - val_loss: 0.7866 - val_accuracy: 0.9375
Epoch 6/30
163/163 [==============================] - 146s 896ms/step - loss: 0.6190 - accuracy: 0.9354 - val_loss: 0.6436 - val_accuracy: 0.9375
Epoch 7/30
163/163 [==============================] - 173s 1s/step - loss: 0.5196 - accuracy: 0.9339 - val_loss: 0.5812 - val_accuracy: 0.8750
Epoch 8/30
163/163 [==============================] - 173s 1s/step - loss: 0.4422 - accuracy: 0.9427 - val_loss: 0.4916 - val_accuracy: 1.0000
Epoch 9/30
163/163 [==============================] - 170s 1s/step - loss: 0.3925 - accuracy: 0.9375 - val_loss: 0.4585 - val_accuracy: 0.8750
Epoch 10/30
163/163 [==============================] - 175s 1s/step - loss: 0.3803 - accuracy: 0.9310 - val_loss: 0.4762 - val_accuracy: 0.8750
Epoch 11/30
pythonProject > modele1_version2.py                    4:37  LF  UTF-8  4 spaces  imenEnvironnementTP1
```

```
Run  modele2_version2
Epoch 2/10
163/163 [==============================] - 260s 2s/step - loss: 0.2768 - accuracy: 0.9331 - val_loss: 1.5913 - val_accuracy: 0.6250
Epoch 3/10
163/163 [==============================] - 256s 2s/step - loss: 0.2506 - accuracy: 0.9473 - val_loss: 1.8399 - val_accuracy: 0.6250
Epoch 4/10
163/163 [==============================] - 251s 2s/step - loss: 0.2307 - accuracy: 0.9515 - val_loss: 1.6337 - val_accuracy: 0.6250

Epoch 00004: ReduceLROnPlateau reducing learning rate to 4.9999998736893760e-06.
Epoch 5/10
163/163 [==============================] - 249s 2s/step - loss: 0.2123 - accuracy: 0.9542 - val_loss: 1.5218 - val_accuracy: 0.6875
Epoch 6/10
163/163 [==============================] - 229s 1s/step - loss: 0.2065 - accuracy: 0.9567 - val_loss: 1.1386 - val_accuracy: 0.6875
Epoch 7/10
163/163 [==============================] - 217s 1s/step - loss: 0.1945 - accuracy: 0.9636 - val_loss: 1.2687 - val_accuracy: 0.6875
Epoch 8/10
163/163 [==============================] - 223s 1s/step - loss: 0.1921 - accuracy: 0.9615 - val_loss: 0.9509 - val_accuracy: 0.6875
Epoch 9/10
163/163 [==============================] - 219s 1s/step - loss: 0.1857 - accuracy: 0.9638 - val_loss: 0.8610 - val_accuracy: 0.7500
Epoch 10/10
163/163 [==============================] - 219s 1s/step - loss: 0.1867 - accuracy: 0.9624 - val_loss: 0.6749 - val_accuracy: 0.7500
Found 624 images belonging to 2 classes.
20/20 [==============================] - 13s 598ms/step - loss: 0.4337 - accuracy: 0.9006
Test Accuracy: 90.06%
Test Loss: 0.4337
/Users/mac/anaconda3/envs/imenEnvironnementTP1/lib/python3.6/site-packages/keras/utils/generic_utils.py:497: CustomMaskWarning: Custom mask layers require a config and
  category=CustomMaskWarning)
pythonProject > modele1_version2.py                    4:37  LF  UTF-8  4 spaces  imenEnvironnementTP1
```



04.

Le modèle MobileNetV2 a montré de très bonnes performances sur la tâche de classification des radiographies thoraciques, avec une **précision de 90,06%** , ce qui est un excellent résultat pour une tâche de classification binaire (normal vs. pneumonie).
La **perte** du modèle est de **0,4337**, ce qui indique que bien que le modèle ait une précision élevée.
Cependant, étant donné la précision élevée, cette perte est relativement faible et suggère que le modèle fonctionne bien.

04. Modèle CNN personnalisé (From Scratch)

Le modèle utilise un réseau de neurones convolutifs (CNN) pour apprendre à partir d'images.
Étapes du modèle :

1.Prétraitement des données :

- Les images sont redimensionnées à 150x150 pixels et normalisées (valeurs des pixels divisées par 255) pour une meilleure convergence.
- Des augmentations de données sont appliquées sur les images d'entraînement pour améliorer la généralisation du modèle (rotations, translations, zooms, flips, etc.).

2.Architecture du modèle :

- Le modèle CNN comporte plusieurs couches successives :
- **Conv2D :** Des couches de convolution pour extraire des caractéristiques à partir des images.
- **BatchNormalization :** Pour normaliser les activations et accélérer l'entraînement.
- **MaxPooling2D :** Pour réduire la dimensionnalité des cartes de caractéristiques.
- **Dropout :** Pour éviter le surapprentissage (overfitting).
- **Flatten :** Pour convertir les données 2D en un vecteur 1D avant la couche dense.
- **Dense :** Une couche entièrement connectée avec une régularisation L2 pour éviter l'overfitting.
- **Activation :** La couche de sortie utilise une activation sigmoid pour effectuer une classification binaire (normal ou pneumonie).

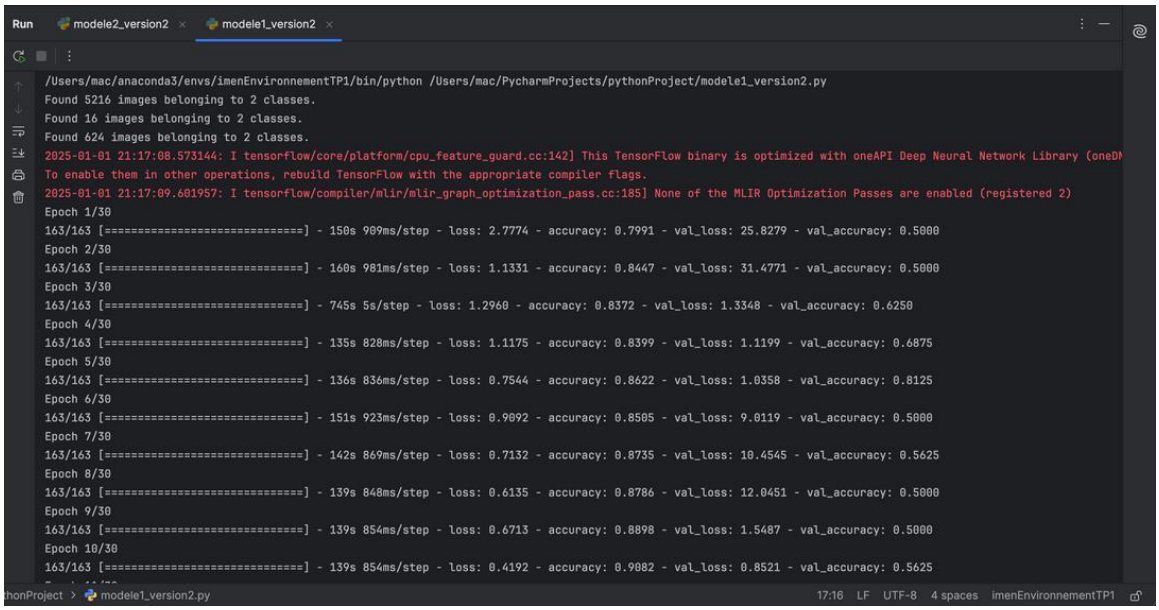
3.Compilation du modèle :

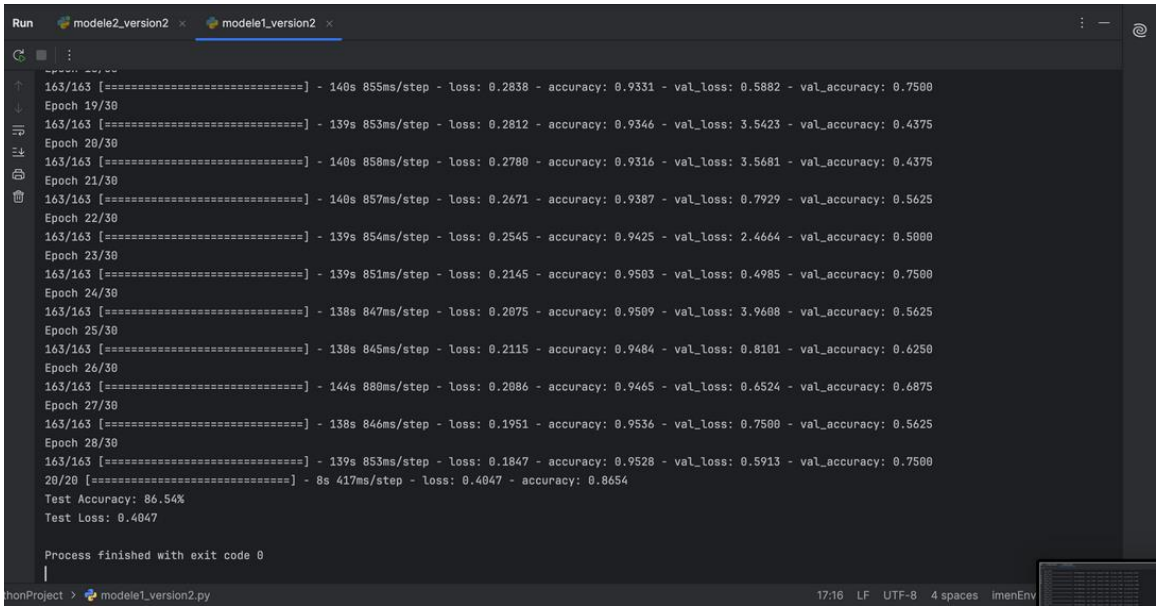
- L'optimiseur Adam est utilisé pour l'optimisation, avec un taux d'apprentissage de 0,001.
- La fonction de perte utilisée est binary_crossentropy, adaptée pour les problèmes de classification binaire.
- Accuracy est utilisée comme métrique pour évaluer les performances du modèle.

4.Entraînement :

- Le modèle est entraîné sur 30 époques, avec des callbacks pour améliorer l'entraînement :
  - **EarlyStopping :** Arrêt anticipé si la perte de validation ne s'améliore pas après 5 époques.
  - **ReduceLROnPlateau :** Réduit le taux d'apprentissage lorsque la perte de validation se stabilise.

Pour ce modèle, on obtient une **précision** (accuracy) de **86,45 %** et une perte (**loss**) de **0,4047**
ce qui indique une bonne capacité de classification tout en laissant une marge d'amélioration, notamment en termes de généralisation et de réduction des erreurs résiduelles.





04. Analyse Comparative des performances des modèles :

Meilleure précision :
Le modèle **MobileNetV2** est le plus performant, avec une précision de 90,06%. Ce modèle est particulièrement adapté pour des tâches de classification d'images tout en étant léger et efficace en termes de ressources.

Modèle robuste mais plus complexe :
DenseNet121 offre une bonne précision de 88,78%, avec une perte plus faible, mais il est plus lourd en termes de calcul et de mémoire, ce qui le rend moins optimal pour des environnements avec des ressources limitées.

Modèle simple mais moins performant :
Le modèle **CNN personnalisé** donne une précision de 86,45% et présente des performances inférieures aux deux autres modèles. Il reste cependant une option valable pour des applications simples ou pour un premier prototype.

	DenseNet121	MobileNetV2	From Scratch
accuracy	88,78%	90,06%	86,45%
loss	0,3912	0,4337	0,4047

05. conclusion :

Dans le cadre de ce projet de détection de la pneumonie à partir de radiographies thoraciques, trois modèles ont été comparés : **DenseNet121, MobileNetV2, et un modèle CNN personnalisé**.
Chacun de ces modèles a montré des performances satisfaisantes, mais avec des différences notables en termes de précision, de perte et d'efficacité.

- **MobileNetV2** s'est distingué comme étant le modèle le plus performant, offrant la meilleure précision (90,06%) tout en étant léger et efficace pour des environnements avec des ressources limitées. Il s'agit d'un excellent choix pour des applications pratiques nécessitant un bon compromis entre performance et efficacité.
- **DenseNet121** a montré des résultats solides avec une précision de 88,78% et une perte de 0,3912, mais sa complexité et sa consommation de ressources plus élevées en font un choix moins optimal pour des systèmes à faible capacité.
- **Le modèle CNN personnalisé** a offert des résultats intéressants avec une précision de 86,45%, mais il reste inférieur aux modèles pré-entraînés, notamment en termes de généralisation et de capacité à traiter des cas complexes.

En conclusion, bien que chaque modèle présente des avantages dans des contextes spécifiques, **MobileNetV2** émerge comme le modèle le plus performant pour cette tâche de classification, grâce à sa haute précision et son efficacité en termes de ressources.
Ce modèle est donc recommandé pour des déploiements pratiques dans des environnements variés, tout en offrant une bonne balance entre performances et consommation de ressources.