

ADVANCED | PROJET

DATA 2

RÉALISÉ PAR :
CHAGOUTI EL HOUCINE
DARBALI OUSSAMA

ENCADRÉ PAR :
Pr. ABDELHAK
MAHMOUDI



Dataset description

Le dataset est couramment connu sous le nom de Chest X-Ray Images (Pneumonia). Il est disponible publiquement la plateformes de données Kaggle.

Le dataset se compose de radiographies thoraciques classées en deux catégories :

- **PNEUMONIA** : Radiographies montrant des signes de pneumonie.
- **NORMAL** : Radiographies montrant des poumons sains, sans signes de pneumonie.

Les images sont organisées en trois répertoires principaux :

1. **train** : Contient les images de formation.
2. **test** : Contient les images de test.
3. **val** (optionnel) : Contient les images de validation.

Simple CNN

Un CNN, ou réseau de neurones convolutifs, est un type de réseau de neurones profond conçu pour traiter des données structurées en grille, comme les images. Les CNN sont inspirés par le cortex visuel des animaux et sont particulièrement efficaces pour reconnaître des motifs dans des images.

Leur architecture se compose principalement de couches de convolution, de couches de pooling et de couches entièrement connectées. Les couches de convolution appliquent des filtres (ou kernels) qui glissent sur l'image d'entrée pour produire des cartes de caractéristiques.

Simple CNN

```
model = Sequential()
model.add(Conv2D(32 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu' , input_shape = (150,150,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.1))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(128 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(256 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 128 , activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 1 , activation = 'sigmoid'))
model.compile(optimizer = "rmsprop" , loss = 'binary_crossentropy' , metrics = ['accuracy'])
model.summary()
```

Evaluation du modèle

```
▶ print("Loss of the model is - " , model.evaluate(x_test,y_test)[0])
print("Accuracy of the model is - " , model.evaluate(x_test,y_test)[1]*100 , "%")

➡ 20/20 [=====] - 19s 960ms/step - loss: 0.2695 - accuracy: 0.9167
Loss of the model is -  0.26950323581695557
20/20 [=====] - 20s 992ms/step - loss: 0.2695 - accuracy: 0.9167
Accuracy of the model is -  91.66666865348816 %
```

le modèle performe bien puisque on a pourcentage de 91% des prédition correcte sur les données de test

Ensemble learning: Bagging

Le Bagging est une technique d'ensemble qui améliore la précision des modèles de prédiction en formant plusieurs modèles sur des échantillons aléatoires de l'ensemble de données d'origine et en combinant leurs prédictions.

Model

```
[ ] def bagging_ensemble(n_members, X_train, y_train, epochs_num):
    ensemble_models = []
    for _ in range(n_members):
        train_ix = np.random.choice(X_train.shape[0], size=X_train.shape[0], replace=True)
        trainX_boot, trainy_boot = X_train[train_ix], y_train[train_ix]

        model = build_cnn_model()
        model.fit(trainX_boot, trainy_boot, epochs=epochs_num, verbose=1)
        ensemble_models.append(model)

    return ensemble_models
```

Evaluation

```
▶ from sklearn.metrics import accuracy_score  
  
evaluate_each_model(ensemble_models,X_test, y_test)  
  
predictions = evaluate_ensemble_models(ensemble_models,X_test, y_test)  
accuracy = accuracy_score(y_test, predictions)  
print(f'Ensemble Accuracy: {accuracy:.3f}')  
  
→ 20/20 [=====] - 2s 89ms/step  
Model 1 Accuracy: 0.881  
20/20 [=====] - 2s 79ms/step  
Model 2 Accuracy: 0.854  
20/20 [=====] - 1s 58ms/step  
Model 3 Accuracy: 0.833  
20/20 [=====] - 1s 54ms/step  
Model 4 Accuracy: 0.856  
20/20 [=====] - 1s 54ms/step  
Model 5 Accuracy: 0.888  
20/20 [=====] - 1s 55ms/step  
20/20 [=====] - 1s 50ms/step  
20/20 [=====] - 1s 28ms/step  
20/20 [=====] - 1s 29ms/step  
20/20 [=====] - 1s 29ms/step  
Ensemble Accuracy: 0.870
```

On voit que le bagging performe un peu moins que le simple CNN

Ensemble learning: Boosting

Le Boosting est une méthode d'ensemble qui améliore la performance des modèles de prédiction en formant des modèles successifs, où chaque modèle corrige les erreurs de son prédecesseur.

Model

```
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier

xgbmodel = XGBClassifier(objective='binary:logistic')
xgbmodel.fit(intermediate_outputs, y_train)
xg_preds_proba = xgbmodel.predict_proba(intermediate_test_output)
xg_preds = (xg_preds_proba[:, 1] > 0.5).astype(int)
```

Evaluation

```
▶ from sklearn.metrics import accuracy_score  
from xgboost import XGBClassifier  
  
xgbmodel = XGBClassifier(objective='binary:logistic')  
xgbmodel.fit(intermediate_outputs, y_train)  
xg_preds_proba = xgbmodel.predict_proba(intermediate_test_output)  
xg_preds = (xg_preds_proba[:, 1] > 0.5).astype(int)  
  
accuracy = accuracy_score(y_test, xg_preds)  
print(f'Model Accuracy: {accuracy:.3f}')
```

le Boosting model a le même accuracy que le bagging

Transfer Learning

Le Transfer Learning est une technique de machine learning où un modèle pré-entraîné sur une tâche est réutilisé comme point de départ pour un modèle sur une autre tâche, ce qui permet de tirer parti des connaissances acquises précédemment.

Model

```
#model = Sequential()
base_model = DenseNet121(include_top=False, weights='imagenet')
x = base_model.output

x = GlobalAveragePooling2D()(x)

predictions = Dense(1, activation="sigmoid")(x)

model = Model(inputs=base_model.input, outputs=predictions)

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Evaluation

```
▶ evaluation = model.evaluate(test)
print(f"Test Accuracy: {evaluation[1] * 100:.2f}%")

evaluation = model.evaluate(train)
print(f"Train Accuracy: {evaluation[1] * 100:.2f}%")

→ 624/624 [=====] - 80s 128ms/step - loss: 0.5889 - accuracy: 0.7949
Test Accuracy: 79.49%
652/652 [=====] - 518s 794ms/step - loss: 0.1637 - accuracy: 0.9421
Train Accuracy: 94.21%
```

Ce modèle est le moins performant parmi les quatre modèles.

Deployment: Streamlit

Streamlit est un framework open-source en Python conçu pour créer des applications web interactives et des visualisations de données. Il permet de transformer des scripts Python en applications web sans nécessiter de compétences en développement web. Grâce à une syntaxe simple, les utilisateurs peuvent ajouter des widgets, des graphiques et des tableaux interactifs.

The screenshot shows a Streamlit application window. At the top right, there are sharing icons: 'Share', a star, a copy symbol, and a more options menu. The main title of the app is 'Prédiction Pneumonie'. Below the title, there is a text instruction 'Choisir une image de radiographie pulmonaire' (Select a chest X-ray image). A central input area contains a cloud icon with an upward arrow, the text 'Drag and drop file here', and a note 'Limit 200MB per file • JPG, JPEG, PNG'. To the right of this input area is a red-outlined button labeled 'Browse files'.

Tester avec une image radiographique d'un patient atteint de pneumonie

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

[Browse files](#)

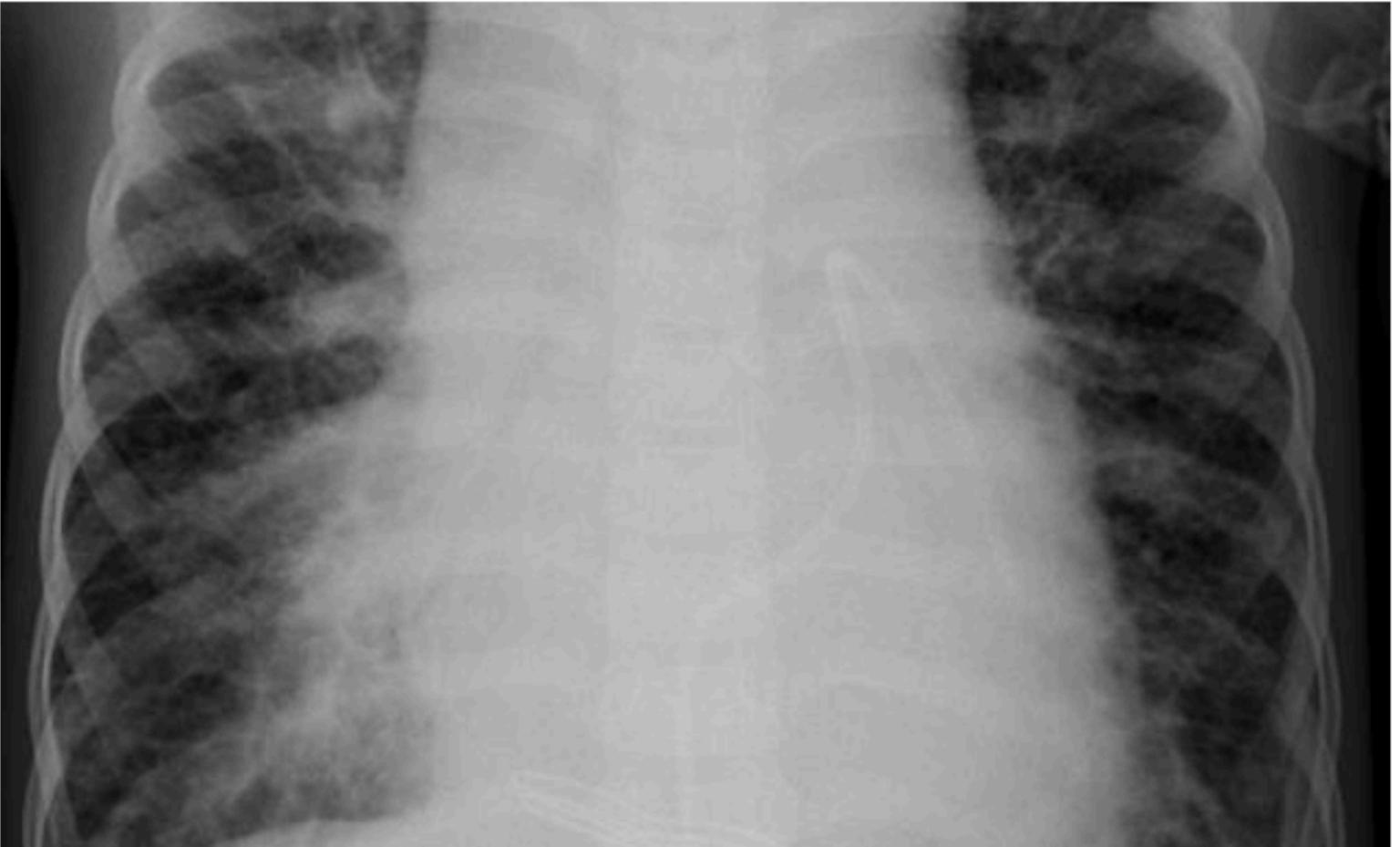
 person1010_virus_1695.jpeg 15.3KB X

Image téléchargée

Prédiction: Malade

[Manage app](#)

What is next ?

- 1. Utiliser des techniques d'augmentation de données pour générer plus d'exemples à partir de l'ensemble de données existant, comme la rotation, le recadrage, le zoom, et la modification de la luminosité des images.**
- 2. Utiliser des techniques de recherche d'hyperparamètres, telles que la recherche en grille (Grid Search) ou la recherche bayésienne, pour optimiser les paramètres du CNN et des modèles de boosting pour améliorer les performances.**
- 3. Utiliser des services de calcul plus puissants et adaptés pour l'entraînement de modèles complexes, tels que les environnements de GPU et TPU sur Google Cloud Platform (GCP), Amazon Web Services (AWS), ou Microsoft Azure, pour réduire les temps d'entraînement et gérer de plus grands ensembles de données.**