

IA DOC

<https://drive.google.com/drive/u/0/folders/15kGUWGDWgzHk0A6t1x7hAkPep95OLaD>
https://github.com/Mulherbe/ia_cnn

Projet de reconnaissance d'image : Détection de pneumonie

Objectif

L'objectif de ce projet est de développer un modèle de reconnaissance d'image pour identifier si des poumons sont sains ou s'ils présentent une pneumonie. Le modèle utilise un réseau neuronal convolutif (CNN) pour analyser les images de poumons et déterminer leur état.

Données

Le modèle a été entraîné et validé sur un jeu de données contenant 7618 images, avec des images de poumons sains et des poumons atteints de pneumonie, ainsi que des images diverses et variées. Le jeu de données est divisé en ensembles d'entraînement, de validation et de test pour évaluer la performance du modèle.

Veille technologique

PyTorch : PyTorch est un framework d'apprentissage automatique open source basé sur Python. Il est largement utilisé pour le développement de modèles d'IA en raison de sa flexibilité, de sa facilité d'utilisation et de sa communauté active. Dans ce projet, PyTorch est utilisé comme framework principal pour la création et l'entraînement du modèle de détection de la pneumonie. Ses fonctionnalités de calcul tensoriel efficace et de gestion des graphiques de calcul sont particulièrement adaptées à ce type de tâche.

Fast.ai : Fast.ai est une bibliothèque d'apprentissage en profondeur haut niveau construite au-dessus de PyTorch. Elle offre une interface simplifiée pour les tâches courantes d'apprentissage automatique, ce qui permet aux développeurs de se concentrer sur la conception du modèle plutôt que sur les détails techniques. Dans ce projet, Fast.ai est utilisé pour simplifier le processus d'entraînement du modèle de détection de la pneumonie, en fournissant des fonctionnalités telles que l'ajustement automatique des hyperparamètres et la gestion des ensembles de données.

Microsoft Azure Notebooks : Microsoft Azure Notebooks est une plateforme cloud qui offre un environnement de développement intégré pour le développement et l'exécution de code Python. Il permet aux chercheurs et aux développeurs d'accéder facilement aux ressources de calcul nécessaires sans avoir à configurer un environnement local complexe. Dans ce projet, Microsoft Azure Notebooks est utilisé pour l'exécution des expériences d'entraînement du modèle, en tirant parti de la puissance de calcul du cloud pour accélérer le processus.

Caffe : Caffe est un framework d'apprentissage en profondeur largement utilisé, principalement pour les applications de vision par ordinateur. Il est connu pour sa vitesse et son efficacité, en particulier lors de l'inférence des modèles. Dans ce projet, Caffe est utilisé pour l'inférence du modèle de détection de la pneumonie après son entraînement. Grâce à ses performances optimisées, il permet une détection rapide et précise de la pneumonie sur les images pulmonaires.

Theano : Theano est une bibliothèque d'apprentissage en profondeur populaire pour Python. Elle permet de définir, d'optimiser et de calculer efficacement des expressions mathématiques impliquant des tableaux multidimensionnels.

Malgré les avantages de ces alternatives, nous avons choisi Python, TensorFlow, Keras et Google Colab pour leur combinaison de fonctionnalités, de performance et de facilité d'utilisation.

Choix des techno 1ère version

Pour la première version du projet, nous sommes partis sur les bibliothèques sklearn, pandas et RandomForestClassifier. Le projet était développé dans un environnement Jupyter Notebook, offrant une interface interactive pour l'exploration et l'analyse des données.

Sklearn :

Scikit-learn, souvent abrégé en sklearn, est une bibliothèque populaire d'apprentissage automatique en Python. Elle propose un large éventail d'outils et d'algorithmes pour le prétraitement des données, l'apprentissage supervisé et non supervisé, la validation croisée, la sélection de modèles, etc. Dans ce projet, sklearn sera utilisé pour entraîner le modèle de classification RandomForestClassifier.

Pandas :

Pandas est une bibliothèque Python destinée à la manipulation et à l'analyse des données. Elle fournit des structures de données et des outils d'analyse performants, tels que les DataFrames, pour faciliter l'importation, la manipulation et la transformation des données. Pandas sera utilisé dans ce projet pour la gestion des données, l'extraction des caractéristiques pertinentes et la préparation des ensembles d'entraînement et de test.

RandomForestClassifier :

RandomForestClassifier est un algorithme d'apprentissage automatique faisant partie de la famille des méthodes d'ensemble, basé sur des arbres de décision. Il combine plusieurs arbres de décision pour former un modèle robuste et précis. Chaque arbre est construit à partir d'un sous-ensemble aléatoire des données d'entraînement et les prédictions finales sont obtenues par vote majoritaire. RandomForestClassifier sera utilisé dans ce projet pour la classification des images de radiographie pulmonaire.

Jupyter Notebook :

Jupyter Notebook est un environnement de développement interactif qui permet l'exécution de code en temps réel, la visualisation des résultats et la création de rapports explicatifs intégrant du code, des graphiques et des textes. Dans ce projet, Jupyter Notebook sera utilisé pour développer le pipeline complet de la détection de pneumonie, en incluant le chargement des données, le prétraitement, l'entraînement du modèle, l'évaluation des performances et la génération de rapports.

Conclusion: ce projet vise à améliorer la détection précoce de la pneumonie pulmonaire et à contribuer aux avancées médicales dans ce domaine

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import numpy as np
```

```
clf = RandomForestClassifier()
```

```
valScore = cross_val_score(
    clf,
    valDataset["samples"],
    valDataset["labels"],
    cv=10
)
```

```
param_grid = {
    'n_estimators': [100, 150, 200],
    'max_depth': [None, None, None]
}
```

```
from sklearn.model_selection import GridSearchCV
```

```
grid_clf = GridSearchCV(clf, param_grid, cv=10)
grid_clf.fit(trainDataset["samples"], trainDataset["labels"])
```

```
gridScore = grid_clf.cv_results_
bestOne = grid_clf.best_estimator_
```

```
clf.base_estimator_ = bestOne
```

```
from sklearn.metrics import accuracy_score
```

```
testPredicted = clf.predict(testDataset["samples"])
accuracyScore = accuracy_score(testDataset["labels"], testPredicted)
```

Choix des techno version 2

Python a été choisi comme langage de programmation en raison de sa syntaxe claire, de sa popularité en science des données et de son écosystème riche en bibliothèques pour la manipulation et l'analyse de données.

TensorFlow a été utilisé comme bibliothèque d'apprentissage automatique pour sa flexibilité, sa compatibilité avec diverses plates-formes matérielles et sa capacité à optimiser les performances des modèles.

Keras, une interface de haut niveau pour TensorFlow, a été sélectionnée pour faciliter la création et l'entraînement de modèles d'apprentissage profond. Grâce à Keras, nous avons pu rapidement définir l'architecture de notre réseau neuronal convolutif (CNN) et utiliser des fonctionnalités intégrées pour le prétraitement des données et l'évaluation des modèles.

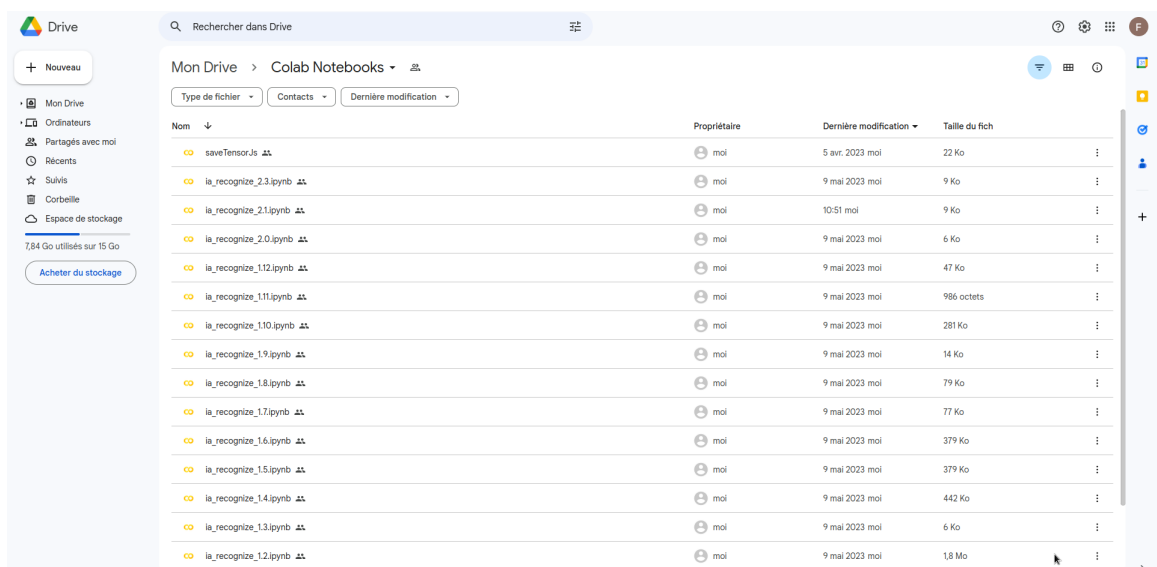
Enfin, nous avons choisi **Google Colab** comme environnement de développement pour tirer parti des ressources de calcul gratuites (GPU et TPU) et faciliter la collaboration entre les membres de l'équipe. Google Colab nous a permis d'entraîner notre modèle.

Organisation

Pour l'organisation de ce projet nous avons décidé d'utiliser google colab et github.

Google colab nous a permis de travailler en équipe et d'utiliser les ressources de Google plutôt que celles de nos pc. Ensuite nous avons stocké nos fichiers ipynb sur le drive google avec un fichier partagé.

Nous avons réalisé beaucoup de test avec différents traitement d'images vu que l'on débute dans le domaine, nous avons créé un fichier pour chaque version, pour garder un historique de toutes les versions. Les versions 1.x sont celles de test, les versions s2.x sont nos versions finales.

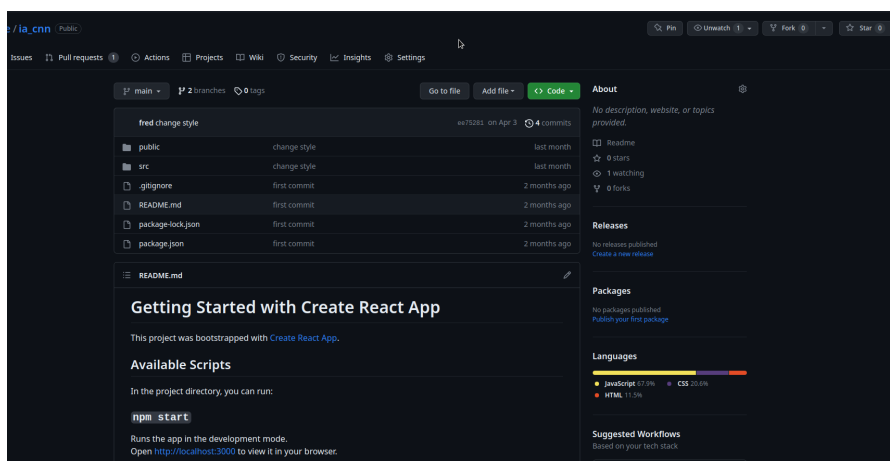


The screenshot shows the Google Drive interface with a list of files under 'Mon Drive > Colab Notebooks'. The files are listed in a table with columns for Name, Owner, Last Modified, and Size. The files are named 'saveTensor.js', 'ia_recognize_2.3.ipynb', 'ia_recognize_2.1.ipynb', 'ia_recognize_2.0.ipynb', 'ia_recognize_1.12.ipynb', 'ia_recognize_1.11.ipynb', 'ia_recognize_1.10.ipynb', 'ia_recognize_1.9.ipynb', 'ia_recognize_1.8.ipynb', 'ia_recognize_1.7.ipynb', 'ia_recognize_1.6.ipynb', 'ia_recognize_1.5.ipynb', 'ia_recognize_1.4.ipynb', 'ia_recognize_1.3.ipynb', and 'ia_recognize_1.2.ipynb'. The sizes range from 6 Ko to 1.8 Mo.

Nom	Propriétaire	Dernière modification	Taille du fich
saveTensor.js	moi	5 avr. 2023	22 Ko
ia_recognize_2.3.ipynb	moi	9 mai 2023	9 Ko
ia_recognize_2.1.ipynb	moi	10-51 mai	9 Ko
ia_recognize_2.0.ipynb	moi	9 mai 2023	6 Ko
ia_recognize_1.12.ipynb	moi	9 mai 2023	47 Ko
ia_recognize_1.11.ipynb	moi	9 mai 2023	986 octets
ia_recognize_1.10.ipynb	moi	9 mai 2023	281 Ko
ia_recognize_1.9.ipynb	moi	9 mai 2023	14 Ko
ia_recognize_1.8.ipynb	moi	9 mai 2023	79 Ko
ia_recognize_1.7.ipynb	moi	9 mai 2023	77 Ko
ia_recognize_1.6.ipynb	moi	9 mai 2023	379 Ko
ia_recognize_1.5.ipynb	moi	9 mai 2023	379 Ko
ia_recognize_1.4.ipynb	moi	9 mai 2023	442 Ko
ia_recognize_1.3.ipynb	moi	9 mai 2023	6 Ko
ia_recognize_1.2.ipynb	moi	9 mai 2023	1,8 Mo

https://github.com/Mulherbe/ia_cnn

Github nous a été utile pour le react que l'on a créé.



Prétraitement des images

Le prétraitement des images a été effectué à l'aide de la classe `ImageDataGenerator` de Keras. Les images ont été redimensionnées à une taille uniforme de 150x150 pixels et leurs valeurs de pixels ont été mises à l'échelle entre 0 et 1 :

```
train_datagen = ImageDataGenerator(rescale=1./255)
```

```
test_datagen = ImageDataGenerator(rescale=1./255)
```

```
val_datagen = ImageDataGenerator(rescale=1./255)
```

Des générateurs d'images ont été créés pour les ensembles d'entraînement, de test et de validation, en utilisant les configurations de prétraitement définies précédemment :

```
train_generator = train_datagen.flow_from_directory(train_dir,  
target_size=(150, 150), batch_size=32, class_mode='categorical',)
```

```
test_generator = test_datagen.flow_from_directory(test_dir,  
target_size=(150, 150), batch_size=32, class_mode="categorical")
```

```
val_generator = val_datagen.flow_from_directory(val_dir,  
target_size=(150, 150), batch_size=32, class_mode="categorical")
```

Architecture du modèle CNN

L'architecture du modèle CNN est définie à l'aide de l'API Keras `Sequential`. Le modèle est composé de trois couches convolutives avec des fonctions d'activation ReLU, suivies de couches de pooling `MaxPooling2D`. Après la dernière couche de pooling, les données sont aplaties (`Flatten`) et passent à travers une couche entièrement connectée (`Dense`) avec 512 neurones et une fonction d'activation ReLU. Une couche de régularisation `Dropout` est ajoutée avant la dernière couche entièrement connectée avec 3 neurones et une fonction d'activation `Softmax` pour la classification multiclasse :

```
model = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(150, 150, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation="relu"),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation="relu"),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation="relu"),
    Dropout(0.5),
    Dense(3, activation="softmax")
])
```


Compilation et entraînement du modèle

Pour éviter le surapprentissage et sélectionner le meilleur modèle possible, un EarlyStopping est utilisé avec un patience de 5 et en restaurant les meilleurs poids :

```
early_stopping = EarlyStopping(monitor='val_loss', patience=5,  
restore_best_weights=True)
```

Le modèle est ensuite entraîné

```
Epoch 1/20  
239/239 [=====] - 83s 300ms/step - loss: 0.2495 - accuracy: 0.9153 - val_loss: 0.2293 - val_accuracy: 0.8846  
Epoch 2/20  
239/239 [=====] - 70s 293ms/step - loss: 0.0933 - accuracy: 0.9664 - val_loss: 0.1284 - val_accuracy: 0.9615  
Epoch 3/20  
239/239 [=====] - 71s 295ms/step - loss: 0.0893 - accuracy: 0.9664 - val_loss: 0.1526 - val_accuracy: 0.9615  
Epoch 4/20  
239/239 [=====] - 71s 296ms/step - loss: 0.0693 - accuracy: 0.9768 - val_loss: 0.3592 - val_accuracy: 0.8077  
Epoch 5/20  
239/239 [=====] - 71s 299ms/step - loss: 0.0403 - accuracy: 0.9857 - val_loss: 0.0680 - val_accuracy: 0.9615  
Epoch 6/20  
239/239 [=====] - 70s 292ms/step - loss: 0.0332 - accuracy: 0.9871 - val_loss: 0.1208 - val_accuracy: 0.9231  
Epoch 7/20  
239/239 [=====] - 70s 294ms/step - loss: 0.0272 - accuracy: 0.9909 - val_loss: 0.0711 - val_accuracy: 0.9231  
Epoch 8/20  
239/239 [=====] - 70s 292ms/step - loss: 0.0328 - accuracy: 0.9882 - val_loss: 0.0526 - val_accuracy: 0.9615  
Epoch 9/20  
239/239 [=====] - 70s 294ms/step - loss: 0.0362 - accuracy: 0.9870 - val_loss: 0.0557 - val_accuracy: 0.9615  
Epoch 10/20  
239/239 [=====] - 70s 294ms/step - loss: 0.0164 - accuracy: 0.9938 - val_loss: 0.2470 - val_accuracy: 0.9615  
Epoch 11/20  
239/239 [=====] - 69s 290ms/step - loss: 0.0167 - accuracy: 0.9925 - val_loss: 0.1129 - val_accuracy: 0.9231  
Epoch 12/20  
239/239 [=====] - 70s 295ms/step - loss: 0.0122 - accuracy: 0.9951 - val_loss: 0.4519 - val_accuracy: 0.9231  
Epoch 13/20  
239/239 [=====] - 70s 294ms/step - loss: 0.0140 - accuracy: 0.9951 - val_loss: 0.1884 - val_accuracy: 0.9615
```

Test

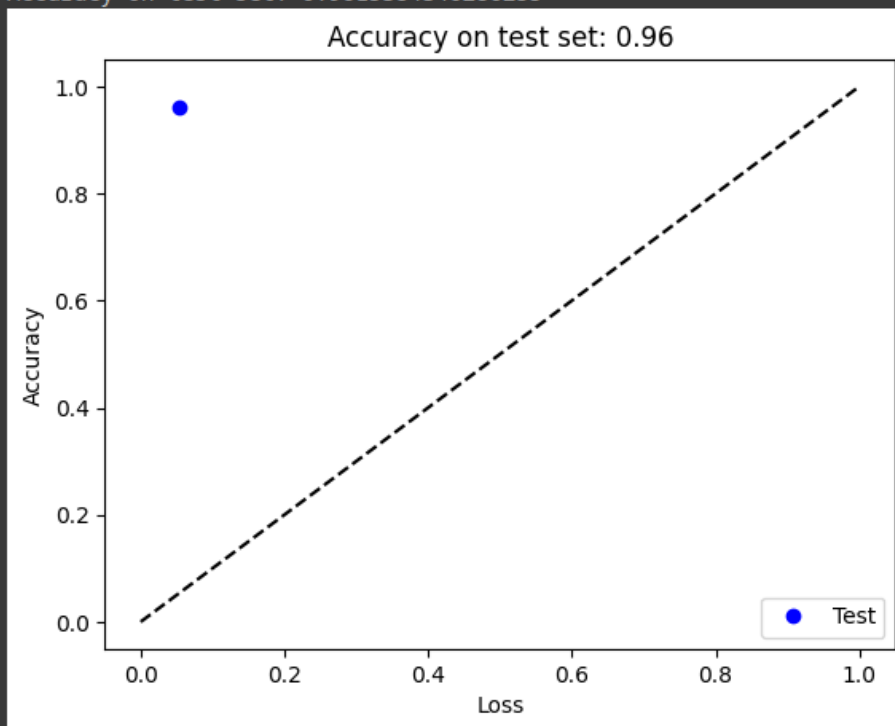
Nous avons réalisé des tests une fois le modèle créé.

```
import matplotlib.pyplot as plt

# Évaluer le modèle sur l'ensemble de test
test_loss, test_acc = model.evaluate(val_generator)
print('Accuracy on test set:', test_acc)

# Créer un graphique pour afficher les résultats
fig, ax = plt.subplots()
ax.plot([0, 1], [0, 1], 'k--')
ax.plot(test_loss, test_acc, 'bo', label='Test')
ax.set_xlabel('Loss')
ax.set_ylabel('Accuracy')
ax.set_title('Accuracy on test set: {:.2f}'.format(test_acc))
ax.legend()
plt.show()
```

```
1/1 [=====] - 0s 197ms/step - loss: 0.0526 - accuracy: 0.9615
Accuracy on test set: 0.9615384340286255
```



[+ Code](#)

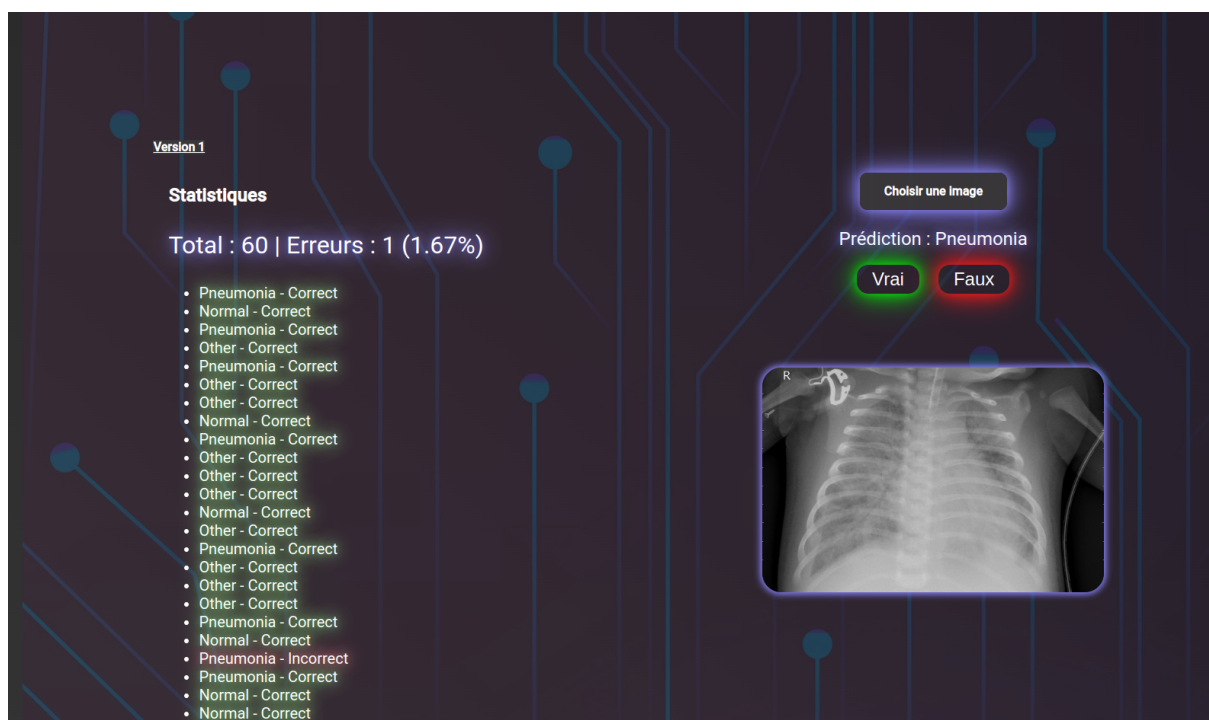
Intégration du modèle dans React avec TensorFlow.js

Après avoir entraîné notre modèle de détection, nous l'avons exporté en TensorFlow.js pour l'intégrer à une application React. Cette étape nous a permis de créer une interface utilisateur conviviale pour que les utilisateurs puissent facilement upload des images de poumons et obtenir des prédictions en temps réel.

En utilisant TensorFlow.js, nous avons pu convertir notre modèle en un format compatible avec le Web et l'intégrer directement dans notre application React. Grâce à cette intégration, notre application peut traiter les images téléchargées par les utilisateurs et fournir des prédictions sur la présence ou l'absence de pneumonie sans avoir à envoyer les données à un serveur distant.

En combinant les avantages de TensorFlow.js et React, nous avons créé une solution Web performante et accessible pour aider à détecter la pneumonie à partir d'images de poumons.

<https://fred.basd2086.odns.fr/>



Axe d'amélioration

MLOps

Une des améliorations possibles était d'automatiser et améliorer l'efficacité du cycle de vie de l'apprentissage automatique:

- Automatisation de l'entraînement des modèles
- Evaluation et validation automatiques
- Déploiement automatisé
- Surveillance de la performance
- Gestion des versions des modèles

Après notre veille nous avons trouvé diverses plateformes qui facilitent le MLOps

- Kubeflow
- Azure Machine Learning
- Google Cloud AI Platform
- Amazon SageMaker

Il en existe bien d'autres. Pour un souci de temps nous n'avons pas eu le temps de mettre en place le MLOps.

API

Notre modèle est actuellement stocké en front, ce qui a ces avantages. Si nous voulions pousser le projet plus loin et réaliser une application mobile ou desktop, une API aurait été plus adaptée pour stocker notre modèle, pour pouvoir faire un seul back et l'utiliser sur différentes plateformes.

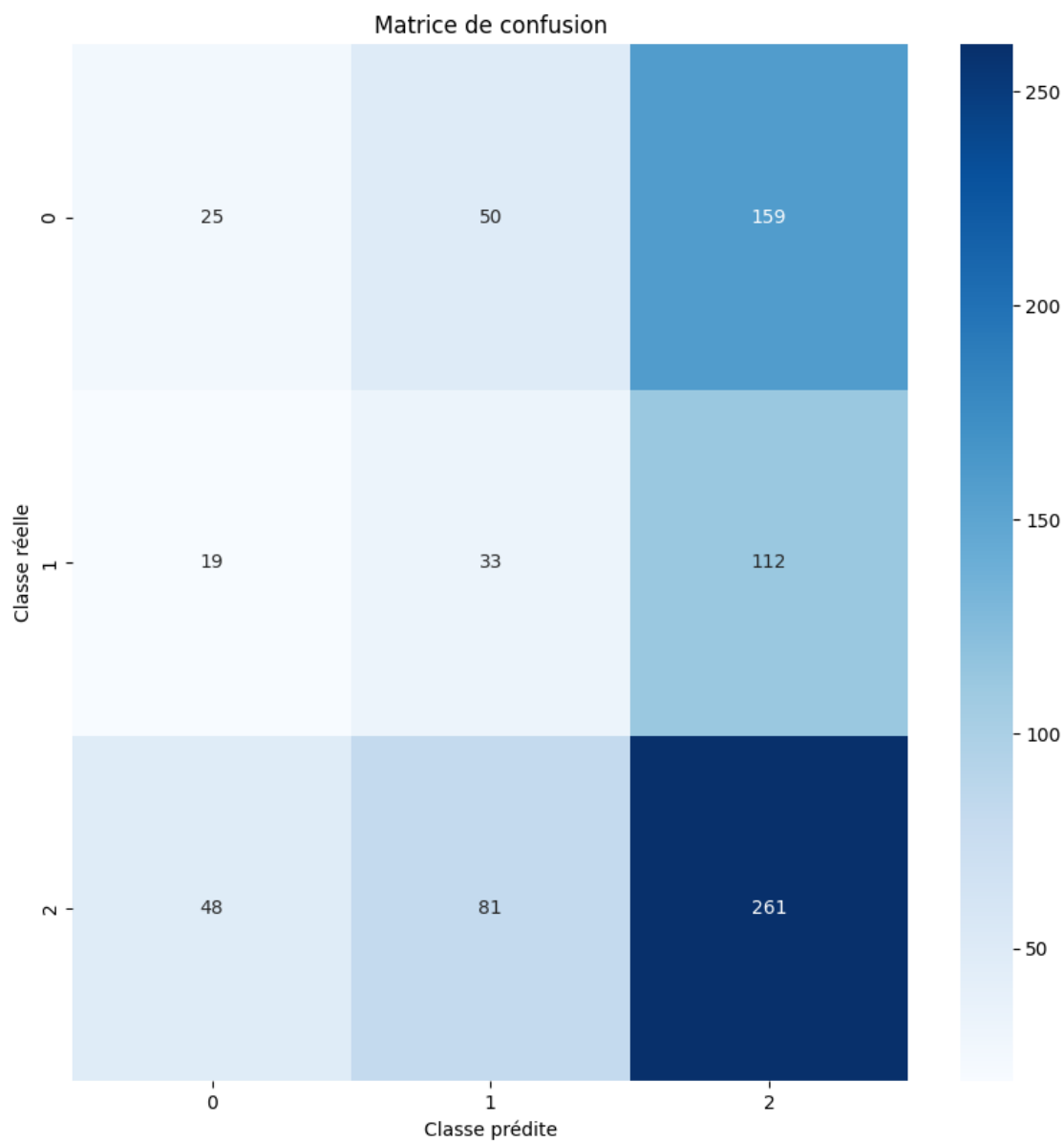
Storage:

Dans l'idée d'enregistrer les images pour les utiliser pour re-entraîner le modèle par la suite, nous aurions pu mettre en place:

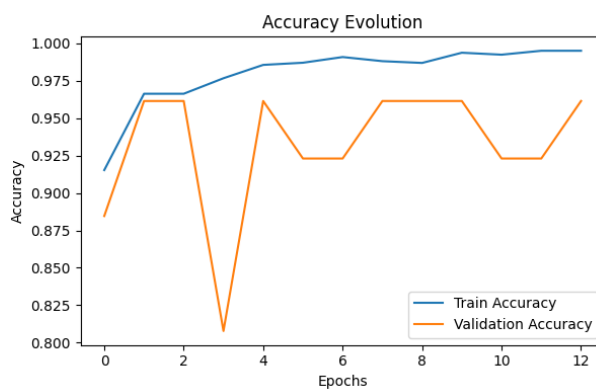
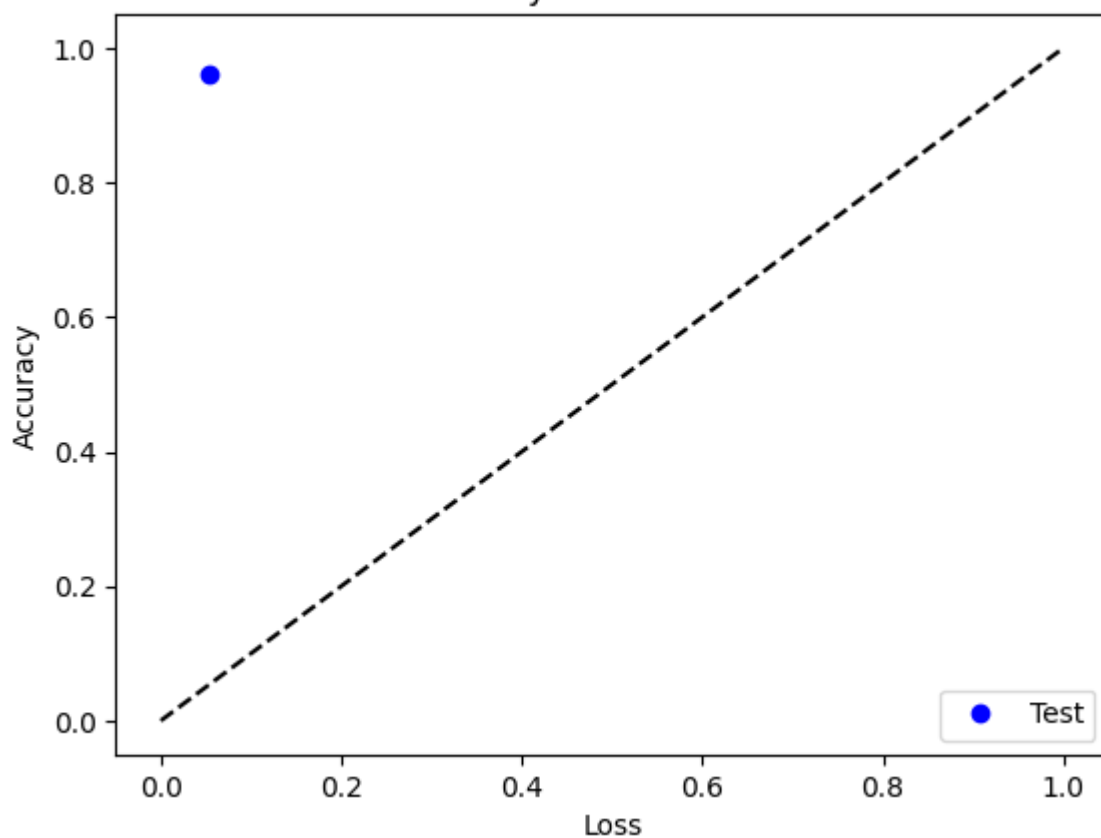
Hot Storage : Ce type de stockage est optimisé pour les données qui doivent être accessibles très rapidement et qui sont fréquemment utilisées. nous pourrions l'utiliser pour les données actuellement utilisées pour l'entraînement et les prédictions,

Cold Storage : Ce type de stockage est optimisé pour les données qui ne sont pas fréquemment utilisées et peuvent être stockées pour de longues périodes. nous pourrions l'utiliser pour archiver les anciens modèles ou les ensembles de données historiques

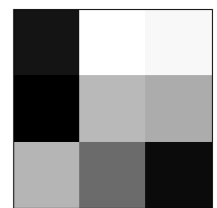
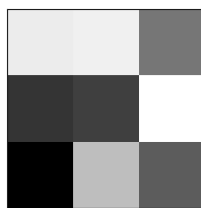
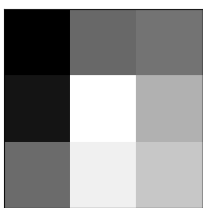
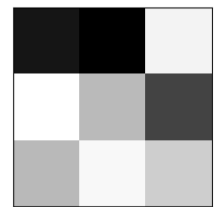
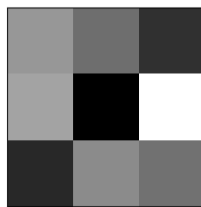
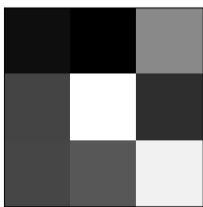
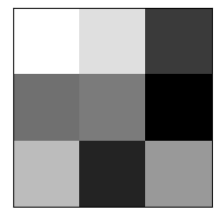
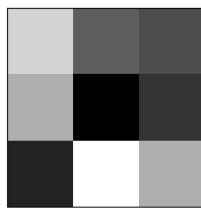
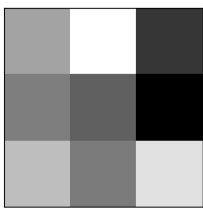
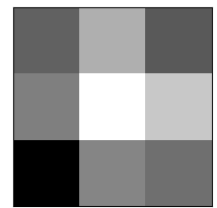
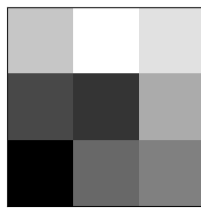
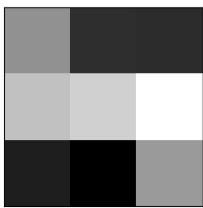
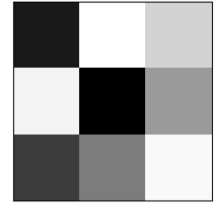
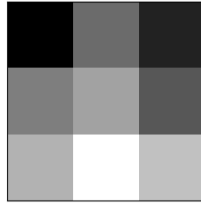
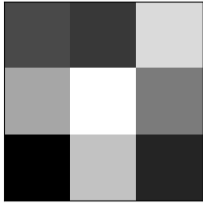
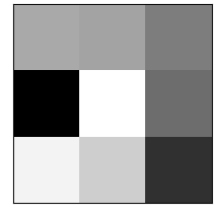
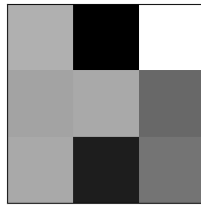
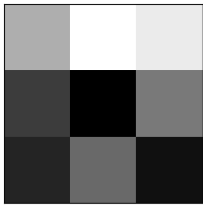
Annexes



Accuracy on test set: 0.96

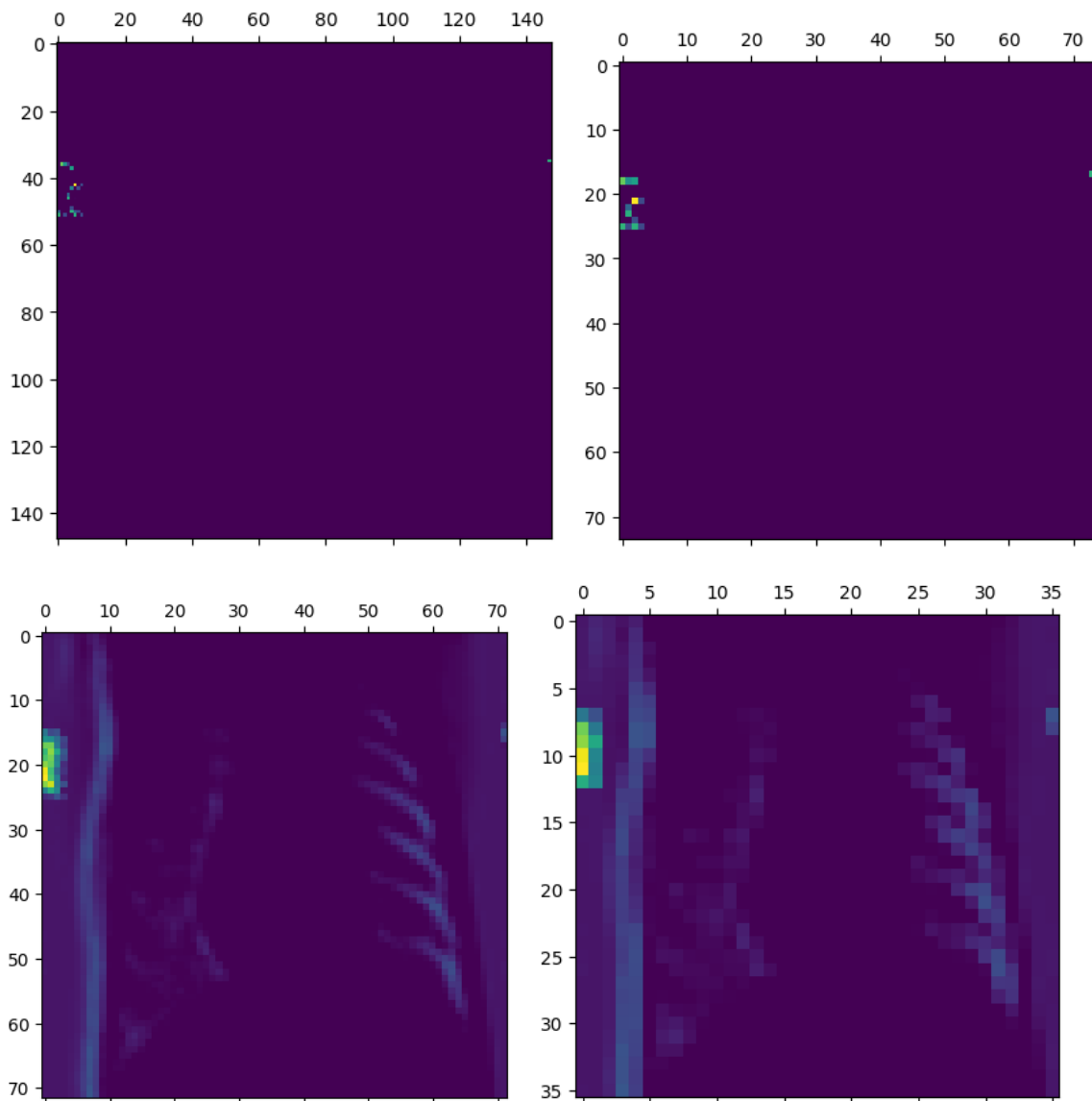


Visualisation des filtres de convolution



Feature maps

Chaque filtre détecte une caractéristique spécifique dans l'image, comme les bords, les coins, ou d'autres types de formes ou de textures.



Clustering des informations entre les classes

