**Analyse de radiographies pulmonaires Covid-19**

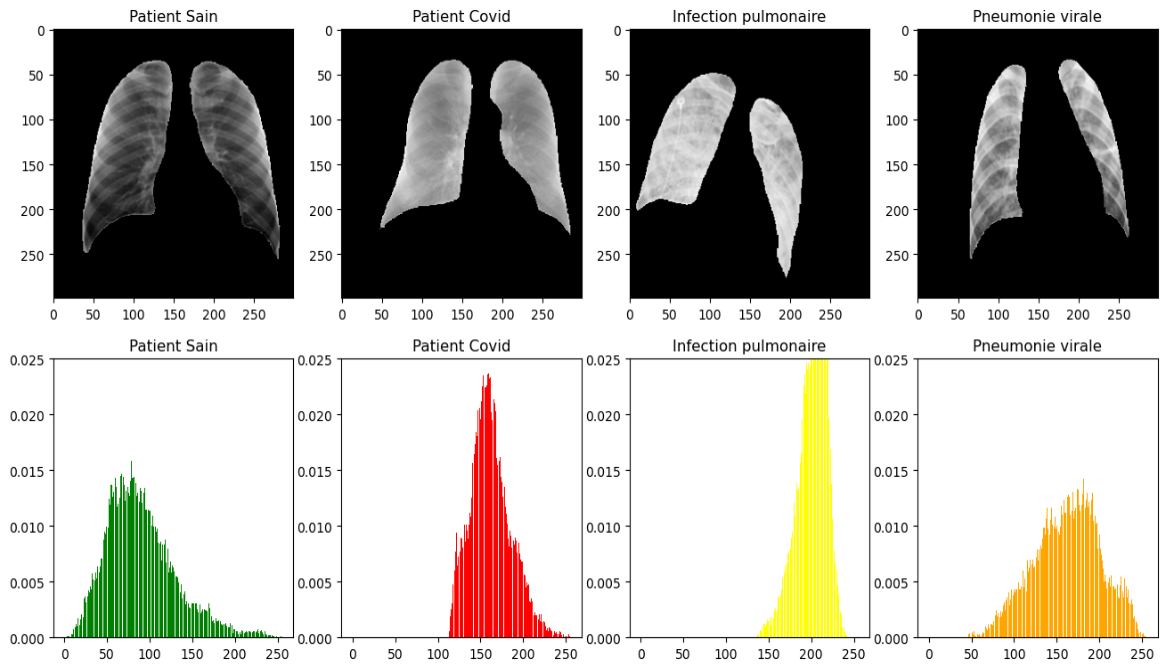
**Rapport de projet n°2 : Modélisation**

**08/12/2023**

## Rappel

Lors de l'exploration de données, nous avons vu que :

* **La surface utile des images est uniquement de 27 % en moyenne**. Nous travaillerons donc sur les images masquées, puisque les masques sont fournis avec le dataset. (Cette approche est recommandée dans plusieurs articles scientifiques consultés, par exemple [celui-ci](https://www.sciencedirect.com/science/article/pii/S0031320322003077), pour s’assurer que le modèle identifie les features à l’intérieur de la surface pulmonaire et non sur le reste de l’image radio).
* Les quantités d'images sont inégales selon la classe de radios. Le jeu de données a été **équilibré** en prenant comme étalon la classe la moins représentée de tel sorte que 20% soit réservé au test et le reste dédié à l'entraînement-validation
* Une réduction de dimension de 256x256 en un simple vecteur 1x256 semble possible en remplaçant l'image par la distribution d'intensité



Nous avons mené l'analyse de deux approches:

1. Une classification DIRECTE des images masquée en 4 classes
2. Une classification RÉDUITE des distributions d'intensité en 2 classes (Normal/Malade)

## Classification du problème

Il s’agit ici d’un problème de classification multi-classe, puisque nous cherchons à déterminer un diagnostic à partir d’images radio, parmi les 4 possibilités suivantes :

Une image contenant capture d’écran

Description générée automatiquement

La tâche à réaliser s’apparente à de la reconnaissance d’image (« computer vision »), avec un enjeu d’extraction des bonnes « features » permettant de caractériser chaque maladie à partir d’une seule image radio. C’est une tâche complexe même pour un ou une radiologie expérimentée : l’interprétation seule de radios du thorax présente une sensibilité (rappel) de 69%[[1]](#footnote-0), donc un taux fort de faux négatifs. En comparaison, le test PCR affiche une sensibilité de 91%.

C’est pourquoi la plupart des organisations professionnelles de radiologie ne recommandent pas l’usage de la radio ou du scanner pour détecter le COVID.

## Choix de la métrique de performance

Pour ce problème, nous avons choisi de comparer les modèles sur la base de l’exactitude, appelé aussi justesse, sur le jeu de validation (« validation accuracy »).

En effet, nous ne cherchons pas seulement à identifier le COVID correctement, mais également à détecter les 3 autres diagnostics possibles (Normal, Pneumopathie virale, Opacité pulmonaire).

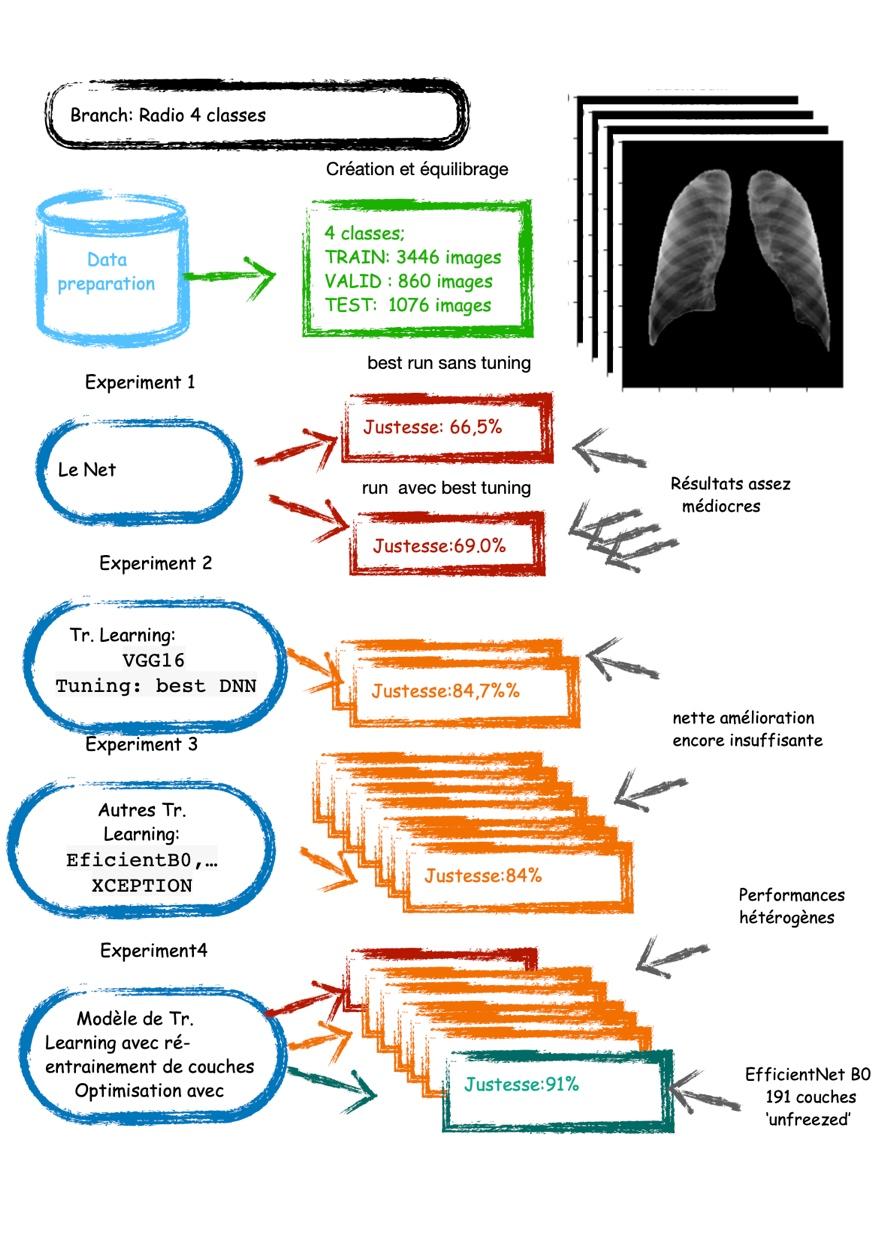
Nous avons équilibré les classes dans le dataset d’entraînement, il y a donc moins de risques que la mesure d’accuracy cache une sur-représentation de la classe « Normale », comme il peut se produire dans un problème de détection d’anomalie.

Une fois les modèles les plus pertinents retenus, nous analyserons la performance sur un jeu de test ainsi que la matrice de confusion et les métriques complètes (f1-score, précision, rappel) pour s’assurer qu’il n’y a pas d’écart majeur de performance entre les classes.

## Choix du modèle et optimisation : approche directe sur la base des images pré-processées

Voici en synthèse les expérimentations successives que nous avons menées :

1. Implémentation du modèle LeNet-5:
   1. Avec différentes tailles de jeu de données, afin d’évaluer la sensibilité du modèle à ce paramètres
   2. Avec Tuning des hyperparamètres
2. Implémentation de transfert en utilisant le modèle VGG16, avec tuning d’hyperparamètres et du meilleur classificateur en sortie
3. Implémentation d'autre d'algorithmes de 'transfert learning'
4. Implémentation de modèles de 'transfert learning' avec réentrainement des couches



Toutes ces expérimentations, à l’exception de la 1.a visant à faire varier les jeux de données, ont été réalisées sur la base d’un **même jeu de données d’entraînement pré-processé comportant 4304 images (= 1076 par classe).**

**Lors de l’import dans Keras, 20% des images sont mises de côté aléatoirement pour la « validation » et le calcul de la « validation accuracy ».**

Le jeu de test utilisé pour l’évaluation finale est également le même pour tous les modèles et correspond à 20% du dataset initial équilibré, soit **1076** images (269 par classe).

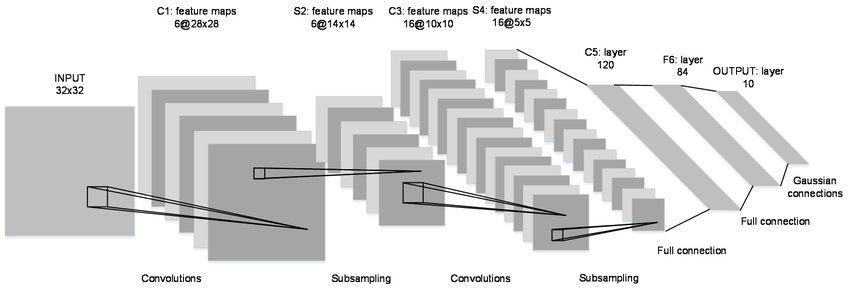
**Synthèse des résultats obtenus :**

| **Run** | **Modèle** | **Caractéristiques** | **Justesse (« val\_acc »)** |
| --- | --- | --- | --- |
| **Exp #1.1** | **LeNet-5** | Sans tuning | **66,5 %** |
| **Exp #1.2** | **LeNet-5** | Avec tuning d’hyperparamètres | **69%** |
| **Exp #2.1** | **VGG16** | FROZEN Sans fine-tuning | **84,7%** |
| **Exp #2.2** | **VGG16** | FROZEN tuning optimisé | **85,6%** |
| **Exp #3.1** | **VGG19** | FROZEN tuning optimisé | **82%** |
| **Exp #3.2** | **XCEPTION** | FROZEN tuning optimisé | **78%** |
| **Exp #3.3** | **ResNet50** | FROZEN tuning optimisé | **80%** |
| **Exp #3.4** | **ResNet101** | FROZEN tuning optimisé | **82%** |
| **Exp #3.5** | **ResNet50V2** | FROZEN tuning optimisé | **81%** |
| **Exp #3.6** | **ResNet101v2** | FROZEN tuning optimisé | **81%** |
| **Exp #3.7** | **DenseNet121** | FROZEN tuning optimisé | **82%** |
| **Exp #3.8** | **DenseNet201** | FROZEN tuning optimisé | **82%** |
| **Exp #3.9** | **MobileNet** | FROZEN tuning optimisé | **83%** |
| **Exp #3.10** | **MobileNetV2** | FROZEN tuning optimisé | **82%** |
| **Exp #3.11** | **EfficientNetB0** | FROZEN tuning optimisé | **84%** |
| **Exp #3.12** | **EfficientNetB7** | FROZEN tuning optimisé | **83%** |
| **Exp #4.1** | **VGG16** | UNFROZEN 4 couches | **71%** |
| **Exp #4.2** | **VGG16** | UNFROZEN 8 couches | **74%** |
| **Exp #4.3** | **VGG16** | UNFROZEN 12 couches | **25%** |
| **Exp #4.4** | **VGG19** | UNFROZEN 5 couches | **25%** |
| **Exp #4.5** | **VGG19** | UNFROZEN 10 couches | **25%** |
| **Exp #4.6** | **XCEPTION** | UNFROZEN 27 couches | **86%** |
| **Exp #4.7** | **XCEPTION** | UNFROZEN 67 couches | **87%** |
| **Exp #4.8** | **XCEPTION** | UNFROZEN 97 couches | **72%** |
| **Exp #4.9** | **ResNet50** | UNFROZEN 32 couches | **74%** |
| **Exp #4.10** | **MobileNet** | UNFROZEN 19 couches | **80%** |
| **Exp #4.11** | **MobileNet** | UNFROZEN 50 couches | **78%** |
| **Exp #4.12** | **EfficientNetB0** | UNFROZEN 75 couches | **85%** |
| **Exp #4.13** | **EfficientNetB0** | UNFROZEN 118 couches | **86%** |
| **Exp #4.14** | **EfficientNetB0** | UNFROZEN 162 couches | **86%** |
| **Exp #4.15** | **EfficientNetB0** | UNFROZEN 191 couches | **91%** |

Détail des expérimentations effectuées

1. **Implémentation du modèle LeNet-5**

Le modèle LeNet-5, développé par Yann LeCun et ses collègues, est un réseau de neurones convolutionnel (CNN) classique. Nous avons utilisé une architecture simple avec des couches de convolution, de sous-échantillonnage, et de couches denses. Le modèle LeNet-5 est connu pour son architecture relativement simple, mais il est efficace pour des tâches de classification d'images simples.



**Représentation du modèle LeNet5 sur la base d’un input 32\*32 -** [**Source**](https://www.researchgate.net/figure/The-LeNet-5-Architecture-a-convolutional-neural-network_fig4_321586653)

1. **Entraînement du modèle LeNet-5 avec différentes tailles de dataset**

Cette expérimentation visait à étudier la sensibilité de la performance du modèle LeNet-5 par rapport à la taille du dataset. Des jeux de données de tailles différentes ont été utilisés, allant de petits ensembles à des ensembles de données plus vastes.

En entraînant le modèle LeNet tuné sur des datasets de taille 1000, 2000, 3000, 4000, 4576 images, nous avons constaté que plus le dataset d’entraînement est important, meilleure est l’accuracy ; en revanche le temps de computation augmente fortement. Cela pourrait justifier le recours à l’augmentation de données afin d’augmenter la performance et réduire l’overfitting. Afin de prioriser l’expérimentation de modèles plus complexes, nous n’avons pas, à ce stade, appliqué d’augmentation d’image.

(Pour rappel, 20% de chaque dataset est dédié au calcul de la « validation accuracy »)

Une image contenant texte, ligne, capture d’écran, Tracé

Description générée automatiquement

1. **Implémentation du modèle LeNet-5 avec tuning des hyperparamètres**

Nous avons effectué un tuning des hyperparamètres pour optimiser les performances du modèle LeNet-5. En utilisant la fonction Keras Tuner, nous avons fait varier les paramètres suivants :

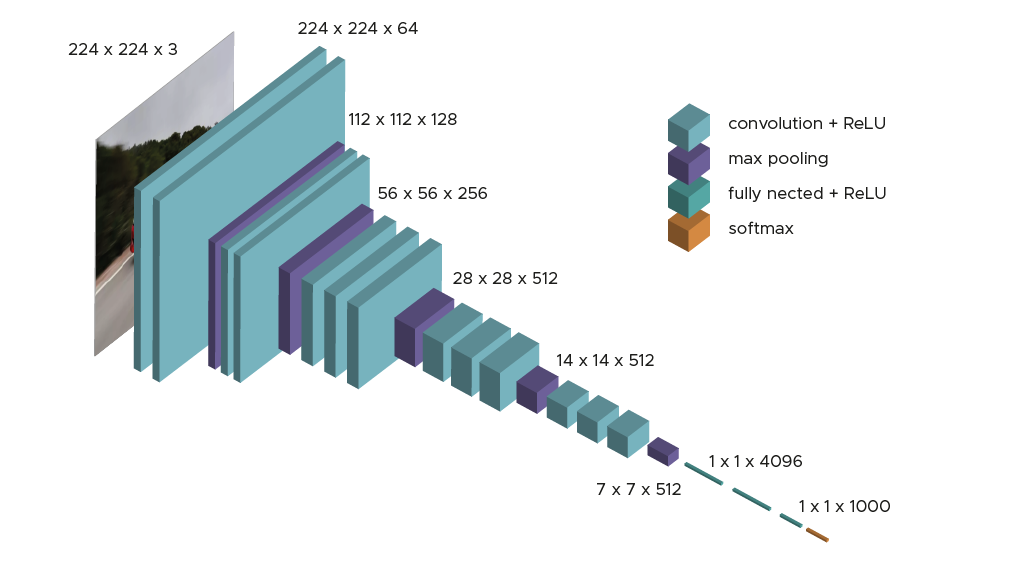
* Nombre de filtres de la première couche de convolution (entre 16 et 128 par pas de 32)
* Nombre de filtres de la deuxième couche de convolution (entre 16 et 128 par pas de 32)
* Intensité du dropout à la sortie des couches de convolution (entre 0 et 0.5 par pas de 0.1)
* Nombre de neurones de la première couche dense (parmi 32, 64, 128, 256)
* Intensité du dropout à la sortie de la couche dense (entre 0 et 0.5 par pas de 0.1)
* Pas d’apprentissage (« learning rate ») pour l’optimiser Adam (parmi 0.1, 0.01, 0.001, 0.0001, 1e-05)

1. **Implémentation d’un modèle de transfer learning utilisant le modèle VGG16**

Le Transfer Learning a été appliqué en utilisant le modèle pré-entraîné VGG16. Nous avons gelé les couches convolutionnelles préexistantes et ajouté des couches denses spécifiques à notre tâche de classification.

VGG16 est un modèle de réseau de neurones convolutionnel qui a été développé par le Visual Graphics Group (VGG) à l'Université d'Oxford. Il a été créé pour participer au concours ImageNet Large Scale Visual Recognition Challenge (ILSVRC) en 2014, où il a obtenu de très bonnes performances.

Le modèle VGG16 est caractérisé par sa profondeur, composée de 16 couches de convolution et de sous-échantillonnage (pooling) suivi de trois couches denses. Les couches de convolution sont toutes configurées avec des filtres de petite taille (3x3) et un pas (stride) de 1, ce qui donne une architecture très uniforme.



**Schéma de la structure de VGG16 – Source** [**Datascientest**](https://datascientest.com/quest-ce-que-le-modele-vgg)

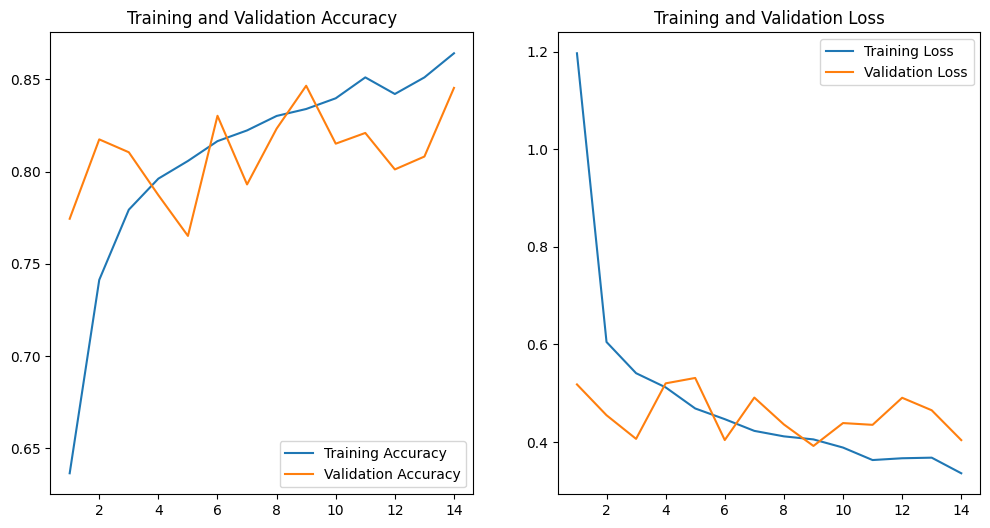
Le Transfer Learning permet de réutiliser un modèle pré-entraîné pour une tâche similaire. Dans le cas de VGG16, le modèle a été pré-entraîné sur la base de données ImageNet pour classifier une large variété d'images. Même si les images n’ont pas de rapport avec les images radio de notre jeu de données, les capacités d’extraction de features de VGG16 en font un candidat intéressant.

Dans un premier temps, nous avons entraîné un modèle composé :

* D’une couche d’entrée permettant d’injecter des images au format (256, 256, 3). Le modèle VGG16 est paramétré pour des images au mode RGB.
* D’une couche de « resizing » permettant de réduire les images en taille (224, 224, 3), qui est le format d’entrée du modèle VGG16
* Du modèle VGG16 importé sans les couches denses de classification
* D’une couche de GlobalAveragePooling pour convertir la sortie du modèle VGG16
* De plusieurs couches denses pour la classification en 4 classes.

La fonction de preprocessing fournie par le package VGG16 de Keras a été appliquée au jeu de données d’entraînement, de validation et de test (centrage des valeurs de pixels notamment), en amont du modèle.

Les couches du modèles VGG16 ont été « gelées » pour pouvoir utiliser les poids liés au pré-entraînement sur le jeu ImageNet. Le modèle a été compilé avec un optimizer de type Adam, la fonction de perte « sparse categorical crossentropy » adaptée à la classification multi-classes, et la mesure d’accuracy.



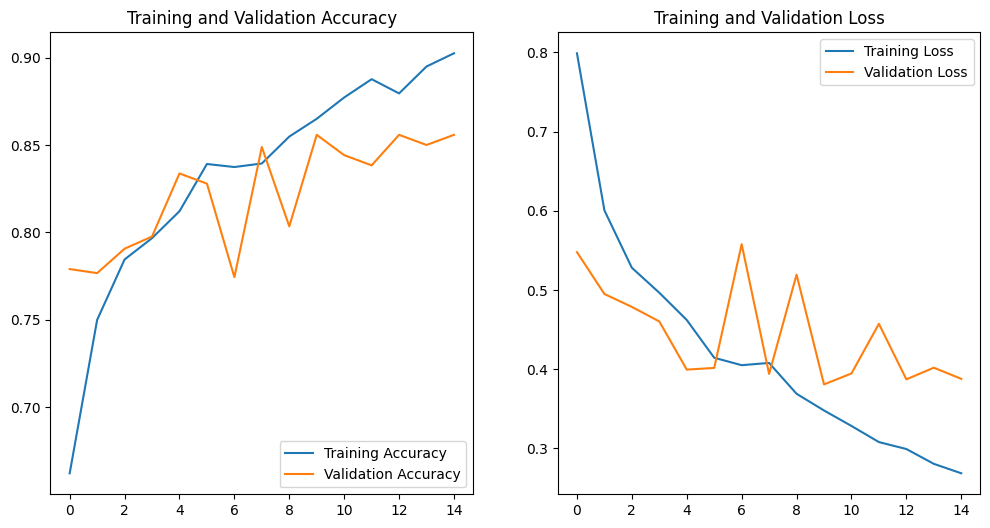
Courbes d’accuracy et loss pour le modèle de transfer learning utilisant VGG16   
(sans finetuning)

**Optimisation du classifieur de sortie du modèle VGG 16**

Nous avons effectué un tuning des hyperparamètres pour optimiser les performances de ce modèle basé sur VGG16. En utilisant le tuner Hyperband du package Keras Tuner, nous avons fait varier les paramètres suivants :

* Intensité du dropout à la sortie du modèle VGG16 (entre 0 et 0.5 par pas de 0.1)
* Nombre de neurones de la première couche dense (parmi 0, 32, 64, 128, 256, 512)
* Intensité du dropout à la sortie de la couche dense (entre 0 et 0.5 par pas de 0.1)
* Nombre de neurones de la deuxième couche dense (parmi 0, 32, 64, 128, 256, 512)
* Pas d’apprentissage (« learning rate ») pour l’optimiser Adam (parmi 0.1, 0.01, 0.001, 0.0001, 1e-05)

Le meilleur modèle a ensuite été implémenté et entraîné sur le jeu de 4503 images, et a atteint une performance de 86% sur les données de validation. Evalué sur le jeu de test, il atteint 83%.



Courbes d’accuracy et loss pour le modèle de transfer learning utilisant VGG16   
(avec optimisation des hyperparamètres)

Le rapport de classification calculé sur le jeu de données de test apporte des précisions : le f1-score global est de 82%, et le modèle performe particulièrement bien sur la classe « Viral Pneumonia ». Des améliorations peuvent être attendues sur les 3 autres classes.

precision recall f1-score support

COVID 0.73 0.81 0.77 269

Lung\_Opacity 0.79 0.74 0.76 269

Normal 0.81 0.77 0.79 269

Viral\_Pneumonia 0.97 0.98 0.97 269

accuracy 0.82 1076

macro avg 0.83 0.82 0.82 1076

weighted avg 0.83 0.82 0.82 1076

Le modèle de classificateur final retenue sera le suivant:

x = Dense(units=128, activation="tanh")(x)

x = Dense(units=32, activation="tanh")(x)

x = Dropout(0.2)(x)

x = Dense(units=16, activation="tanh")(x)

Celui-ci sera utilisé en sortie des expérimentations suivantes.

1. **Implémentation d’autres modèles de transfer learning:**

L'expérimentation suivante a consisté à tester d'autres modèles de transfert learning existant pour voir si l'un d'eux se dégager.

Ont été essayé les modèles suivants:

* VGG19
* XCEPTION
* ResNet50
* ResNet101
* ResNet50V2
* ResNet101v2
* DenseNet121
* DenseNet201
* MobileNet
* MobileNetV2
* EfficientNetB0
* EfficientNetB7

1. **Réentrainement de couches de transfert learning**

Comme l'utilisation des modèles existant de 'transfert learning' tel quel, ne les démarquent pas franchement, nous allons effectuer des réentrainements de couches de ces modèles, en espérant améliorer leur adaptation à notre problématique, et ainsi améliorer les performances.

Les combinaisons suivantes ont été essayées :

* VGG16 avec un réentrainement sur 4,8 et 12 couches
* VGG19 avec un réentrainement sur 5 et 10 couches
* XCEPTION avec un réentrainement sur 27, 67 et 97 couches
* ResNet50 avec un réentrainement sur 32 couches
* MobileNet avec un réentrainement sur 19 et 50 couches
* EfficientNetB0 avec un réentrainement sur 19 et 50 couches

## Choix du modèle et optimisation : approche réduite sur la base des distributions d’intensité

## *a- Classification du problème*

De même que la classification directe des images, la classification de la réduction d'image réduites à leur distribution d'intensité est complexe et a été expérimentée comme une branche parallèle de nos modélisations.

Nous avons tout d’abord expérimenté un modèle de classification simple sur la base des vecteurs d’intensité des images. Comme attendu, la performance a été aussi bonne que 'pile ou face'.

Ce problème réduit à un choix binaire Malade/Pas Malades est typiquement un problème de diagnostic.

Cependant selon l'usage médical, il existe deux situations de diagnostic qui nécessite de privilégier la performance selon certaines métriques :

* le diagnostic d'inclusion: privilégie la justesse ('accuracy') et la spécificité
* le diagnostic d'exclusion privilégie la justesse ('accuracy') et la sensibilité

La métrique de performance principale de cette classification binaire sera la justesse, mais nous considèrerons également la sensibilité et spécificité dans un deuxième temps

Cette approche malade/normal, nous permettra de disposer d'un jeu de données plus larges. Nous disposons ainsi d'un jeu de 5164 images d'entrainements, 1292 pour la validation et enfin 1614 pour le jeu de test

## *b- Choix du modèle et optimisation*

1. Expérimentation #1: KNN, SMV, RandomForest.  Justesse < 60%
2. Expérimentation #2: Deep Learning. Réseau type Le net en dimension 1.
3. Expérimentation #3: Affinage d'une Le Net avec Keras Tuner
4. Optimisation de l'architecture et des paramètres avec Keras Tuner

*Par manque de temps nous avons stopper cette branche d'exploration à ce niveau.*

*Cependant d'autres expérimentation pour l'améliorations du modèle sont possibles tel que:*

1. *Expérimentation #4: Transfert learning?*

*Recherche d'un TF applicable en 1D de traitement du signal*

1. *Expérimentation #5; UnFreeze TF*
2. *Combinaison à d'autres question spécifique*Une image contenant texte, diagramme, Police, ligne

   Description générée automatiquement

## *c- Interprétation des résultats*Une image contenant reçu, texte Description générée automatiquement

L'amélioration du modèle au sein de chaque expérimentation est souvent venue de l'utilisation des bons paramètres dans Keras, et surtout de l'utilisation de Keras Tuner qui nous a permis de sélectionner la bonne architecture présenter à gauche

Les résultats de l'experimentation 3 montre un entrainement optimisé du modèle :Une image contenant Tracé, capture d’écran, diagramme, ligne

Description générée automatiquement

mais également une excellente performance en sensibilité de 94,5% tout en conservant une justesse acceptable de 80%.

|  | pred  MALADE | pred  NORMAL |
| --- | --- | --- |
| MALADE réel | VP:764 | FN:43 |
| NORMAL REEL | FP:279 | VN:528 |

Avec un faible taux de faux négatif, soit une bonne sensibilité

Le modèle devient acceptable pour à l'intégrer dans un protocole d'exclusion. Cependant nous pouvons et devons encore l'améliorer

1. Source : 1. Wong HYF, Lam HYS, Fong AH, et al. Frequency and distribution of chest radiographic ﬁndings in COVID-19 positive patients [published online ahead of print, 2019 Mar 27]. Radiology 2019;201160.doi:10.1148/radiol.2020201160 [↑](#footnote-ref-0)