

# **Projet Deep Learning**

## **Analyse de radiographies pulmonaires Covid-19**

Membres du projet : PAGESY Thomas, LAROSE Jérôme.

Mentor : BENTOUMI Okba.

# Sommaire

- **Introduction**
  1. Contexte
  2. Objectifs
- **Compréhension et manipulation des données**
  1. Cadre
  2. Pertinence
  3. Pre-processing et feature engineering
  4. Gestion des labels et de l'équilibre du dataset
- **Modélisation**
  1. Choix du modèle et motivations
  2. Segmentation d'images
  3. Classification des images masquées
  4. Visualisation des zones d'intérêt avec GradCam
- **Conclusion**
  1. Bilan et interprétation des résultats
  2. Améliorations possibles et autres pistes de réflexion
  3. Remerciements

# Introduction

## 1) Contexte

La radiographie pulmonaire s'est révélée essentielle dans la détection de cas positifs au Covid-19. Cette technique consiste à faire une impression des différences de densité du poumon sur un film radiographique. Elle permet d'étudier les poumons, la trachée, les bronches, le cœur, les vertèbres et les côtes. Initialement, elle était utilisée dans le domaine médical afin de détecter des infections pulmonaires, des cancers, des inflammations ou tout autre type d'anomalie.

Ajoutant maintenant la détection du Covid-19 à son expertise, nous avons cherché à savoir s'il était possible de détecter les cas positifs au Covid-19 à l'aide du deep learning. Pour se faire, nous allons essayer de déterminer un modèle de prédiction sur une base de données contenant des radiographies pulmonaires afin de classifier les cas Covid-19.

Cela pourrait notamment permettre d'utiliser cette méthode dans les hôpitaux et cliniques afin de soulager la charge de travail des médecins sans pour autant négliger l'avis médical d'un professionnel.

## 2) Objectifs

Les principaux objectifs du projet sont :

- L'exploration des données grâce à la visualisation de celles-ci et l'analyse des données.
- La normalisation des données
- L'implémentation d'une architecture U-NET pour la segmentation des images.
- L'implémentation d'une architecture CNN pour la classification des images.
- L'analyse des résultats obtenus et des performances des modèles.
- Éventuellement un processus de visualisation des zones d'intérêt sur les radios classées comme cas positif de covid-19
- Une démonstration via un streamlit.

Il est tout à fait possible d'apporter un avis médical au projet, notamment sur la lecture de radiographie pulmonaire, grâce à l'intervention d'un professionnel du milieu médical. Cela apporterait une expertise supplémentaire sur la détection de différentes anomalies pulmonaires.

# Compréhension et manipulation des données

## 1) Cadre

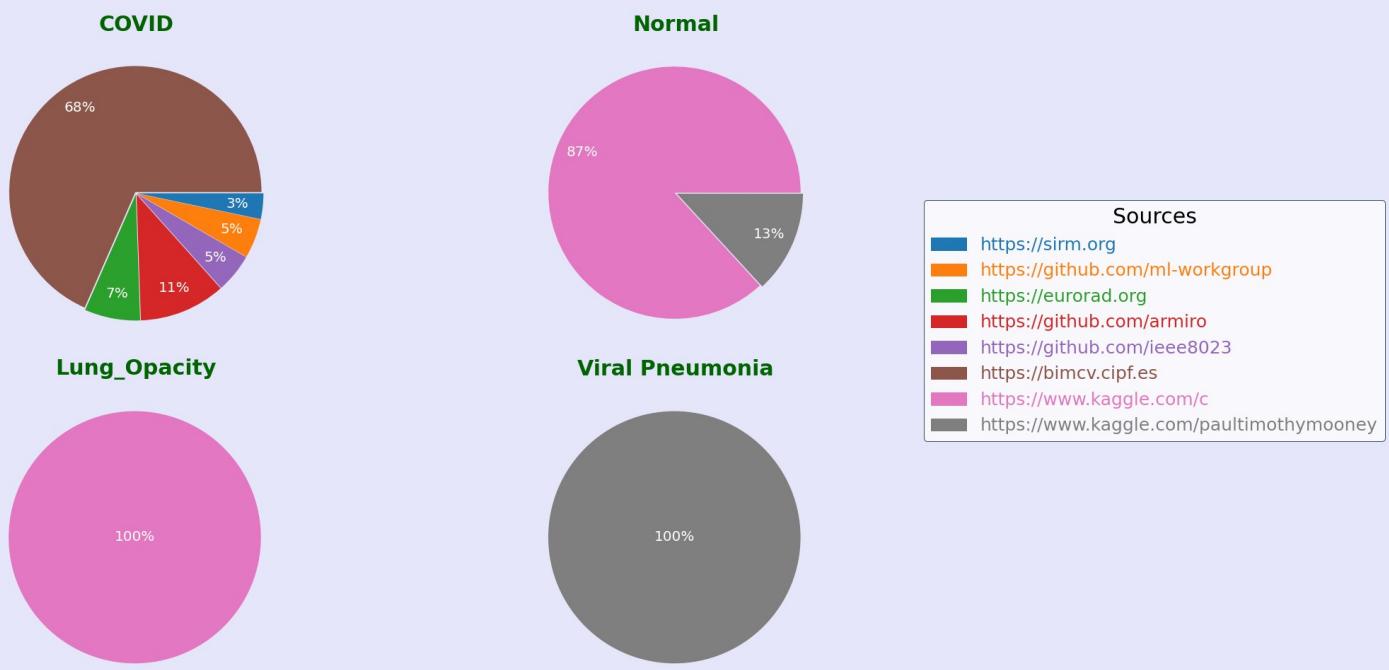
Pour atteindre les différents objectifs du projet, nous avons à disposition une base de données provenant de plusieurs sources :

- <https://sirm.org/category/senza-categoria/covid-19/>
- <https://github.com/ml-workgroup/covid-19-image-repository/tree/master/png>
- <https://eurorad.org>
- <https://github.com/armiro/COVID-CXNet>
- <https://github.com/ieee8023/covid-chestxray-dataset>
- <https://bimcv.cipf.es/bimcv-projects/bimcv-covid19/#1590858128006-9e640421-6711>
- <https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data>
- <https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia>

Parmi ces images, il y a 4 types de radiographie : COVID, Normal, Lung Opacity et Viral Pneumonia correspondant respectivement à cas positifs au Covid-19, poumons normaux, poumons atteints d'opacité pulmonaire et des poumons atteints de pneumonie aiguë.

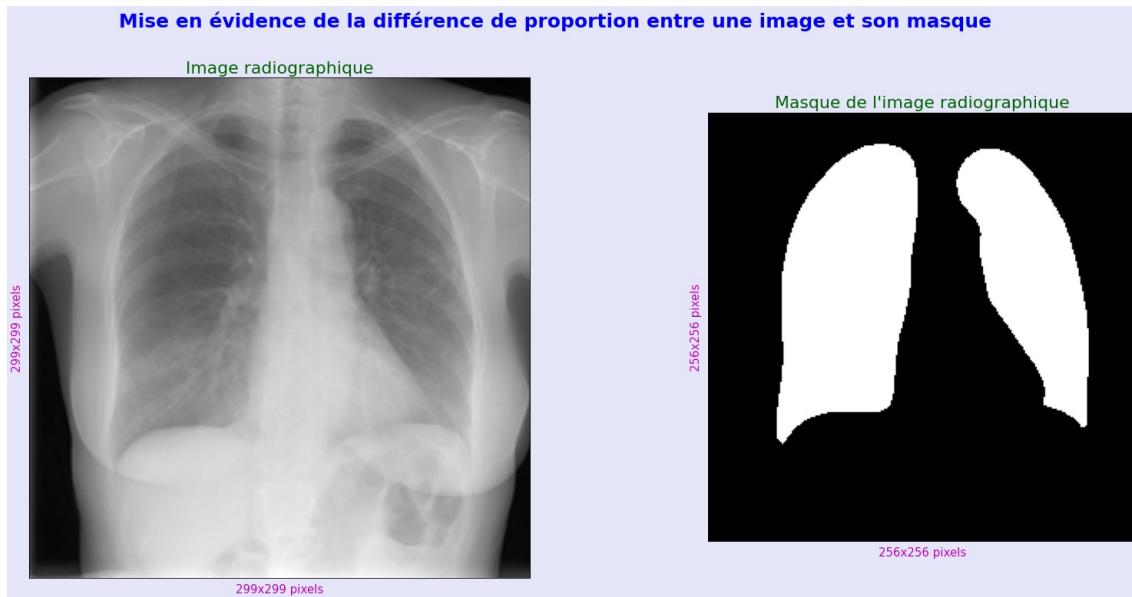
Au total il y a 21165 images, ainsi que leurs masques associés. Parmi elles, 3616 Covid, 10192 Normales, 6012 Lung Opacity et 1345 Pneumonies aiguës. On peut voir ainsi que les proportions des catégories sont différentes selon les sources :

### Proportion des catégories d'images en fonction de leurs source



Chaque image possède son masque respectif, cependant ils n'ont pas les mêmes dimensions :

- Image 299x299 pixels
- Masque 256x256 pixels

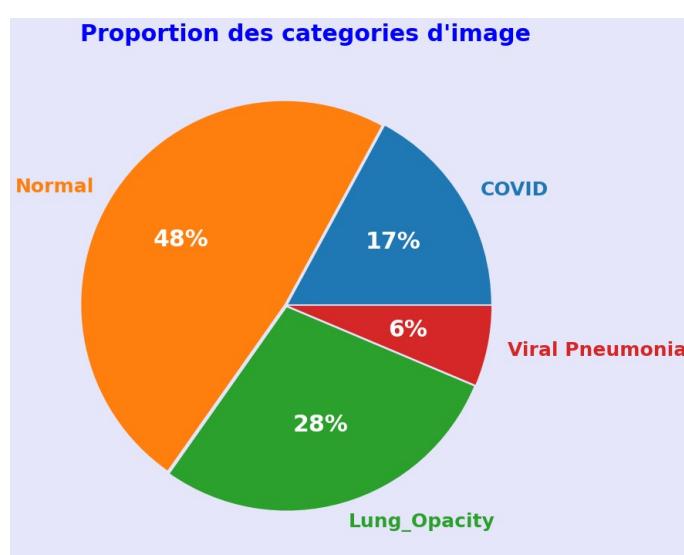


## 2) Pertinence

Les variables principales de cette base de données sont donc les différentes images et les masques correspondant aux 4 catégories citées précédemment. La variable cible correspond aux labels des différentes radiographies, ainsi que leurs masques associés, c'est à dire : le diagnostic. Le jeu de données n'étant que des images avec les masques, les sources et les diagnostiques correspondants, il n'y a pas à proprement parler beaucoup de variété dans les données statistiques. Nous allons tout de même exploiter les différentes informations mises à notre disposition.

## 3) Pre-processing et feature engineering

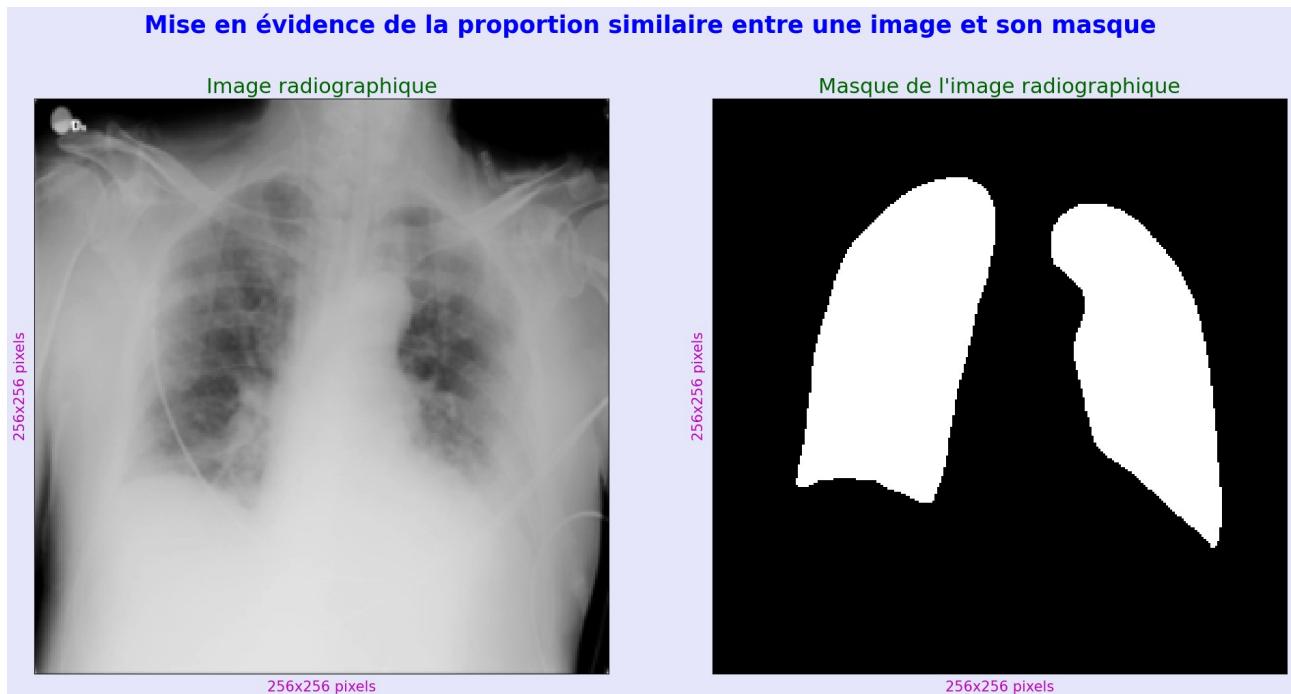
Nous pouvons constater que les classes sont déséquilibrées au sein du dataset :



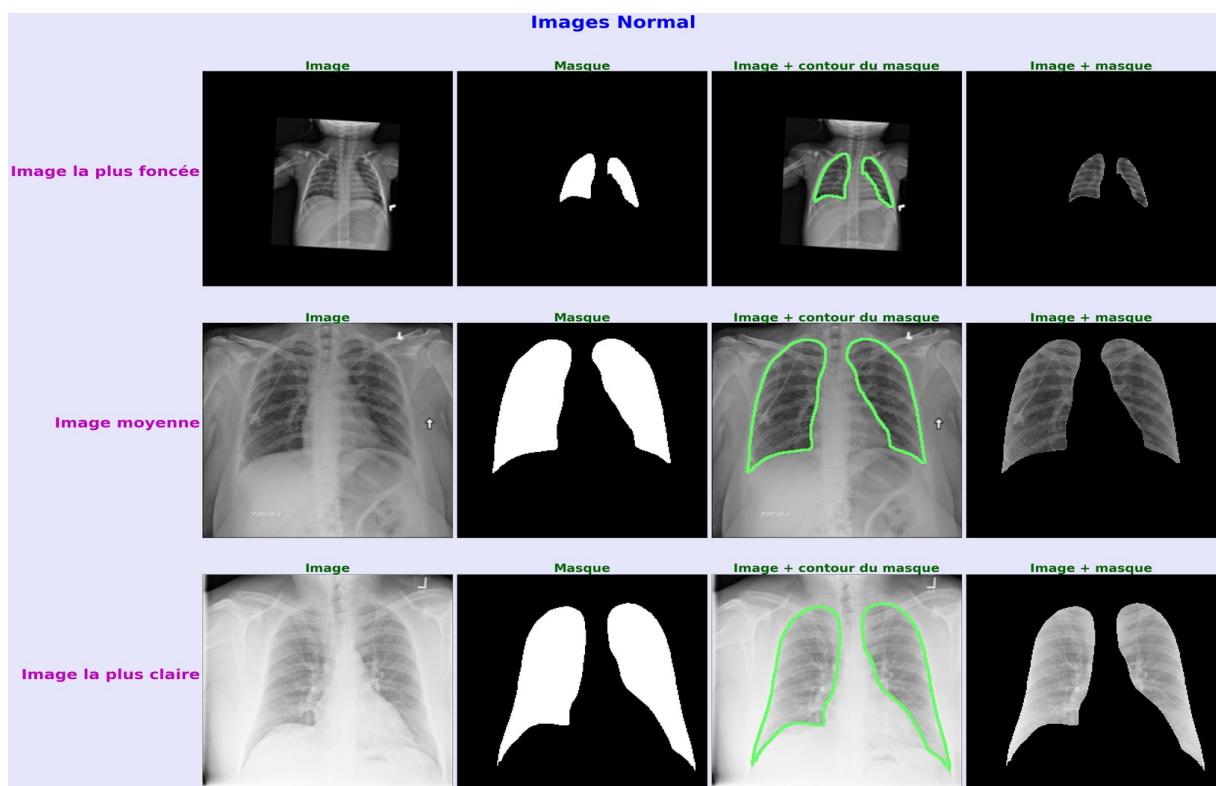
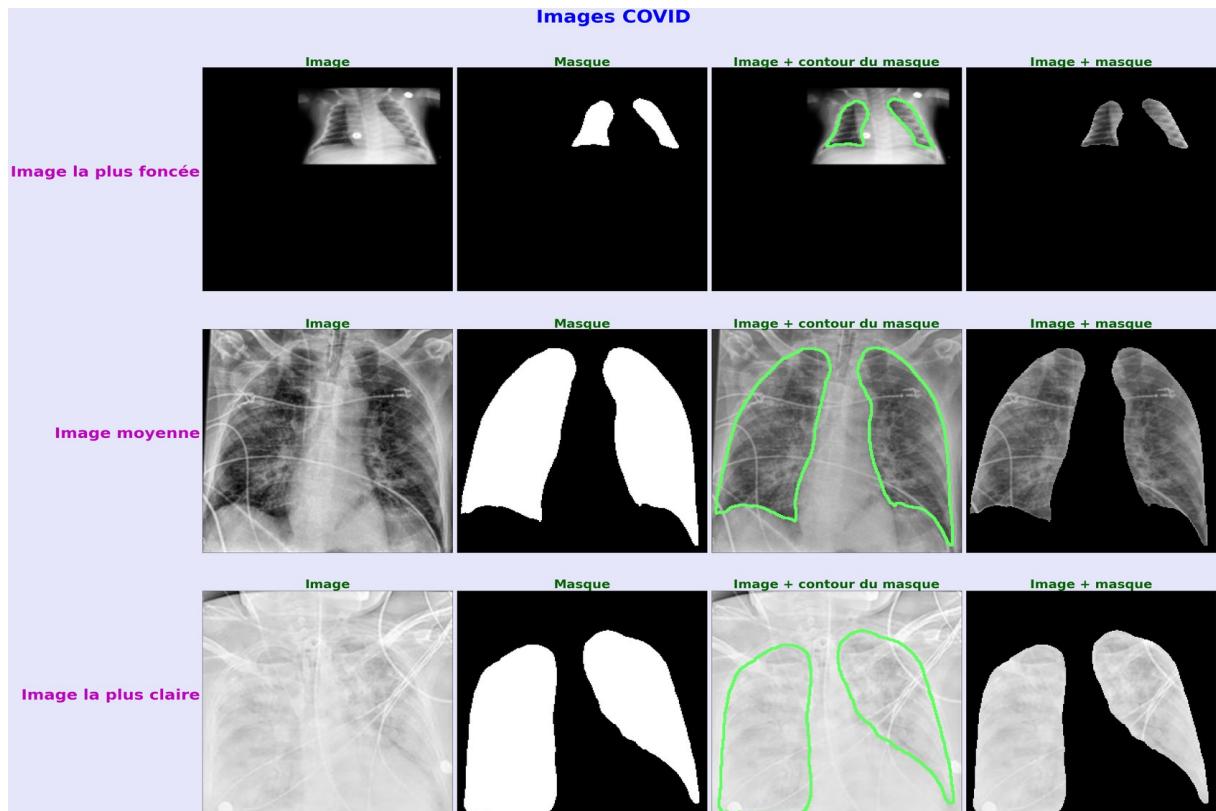
En effet, comme dit précédemment, il y a 21165 images. Parmi elles, 3616 Covid, 10192 Normales, 6012 Lung Opacity et 1345 Pneumonies aiguës.

Cela pourra s'avérer être un problème lors de l'entraînement d'un modèle de prédiction. Nous avons commencé par redimensionner les images pour qu'elles correspondent aux masques :

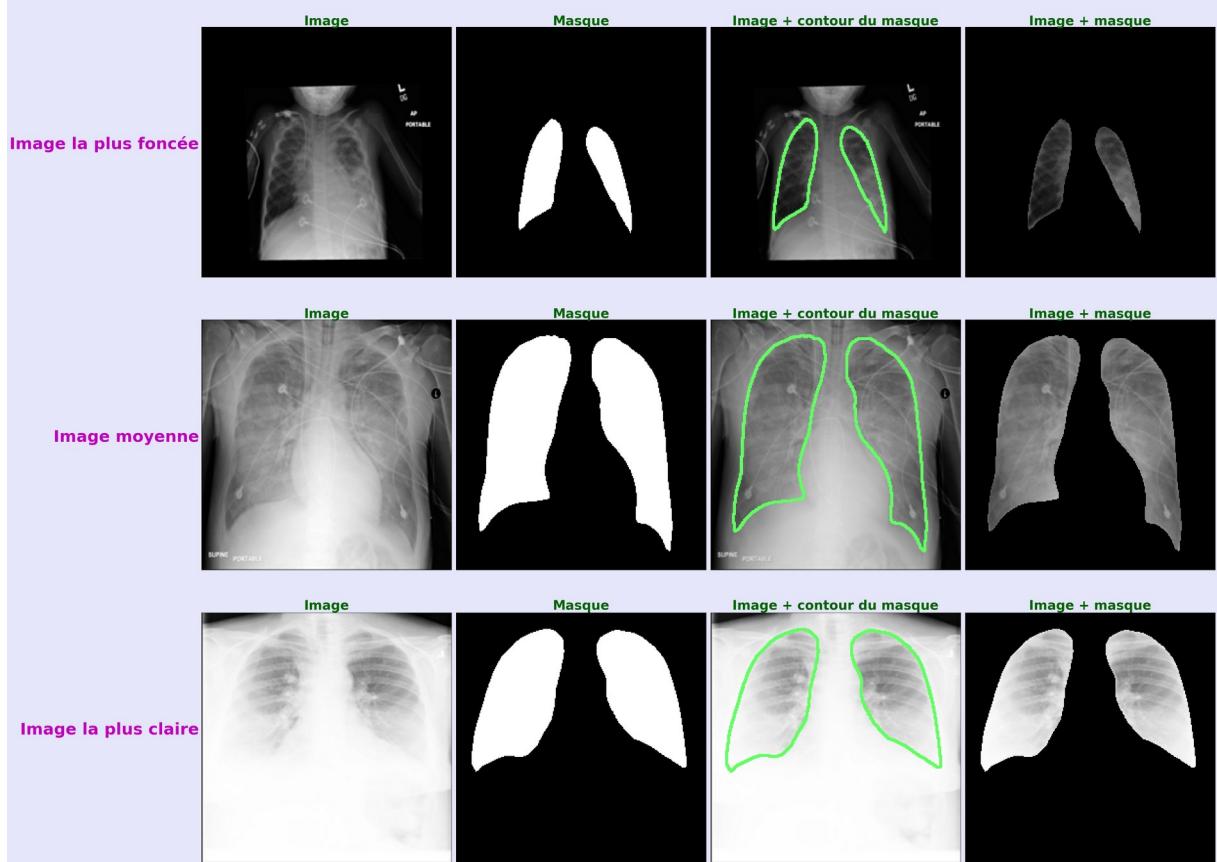
- Image 256x256 pixels
- Masque 256x256 pixels



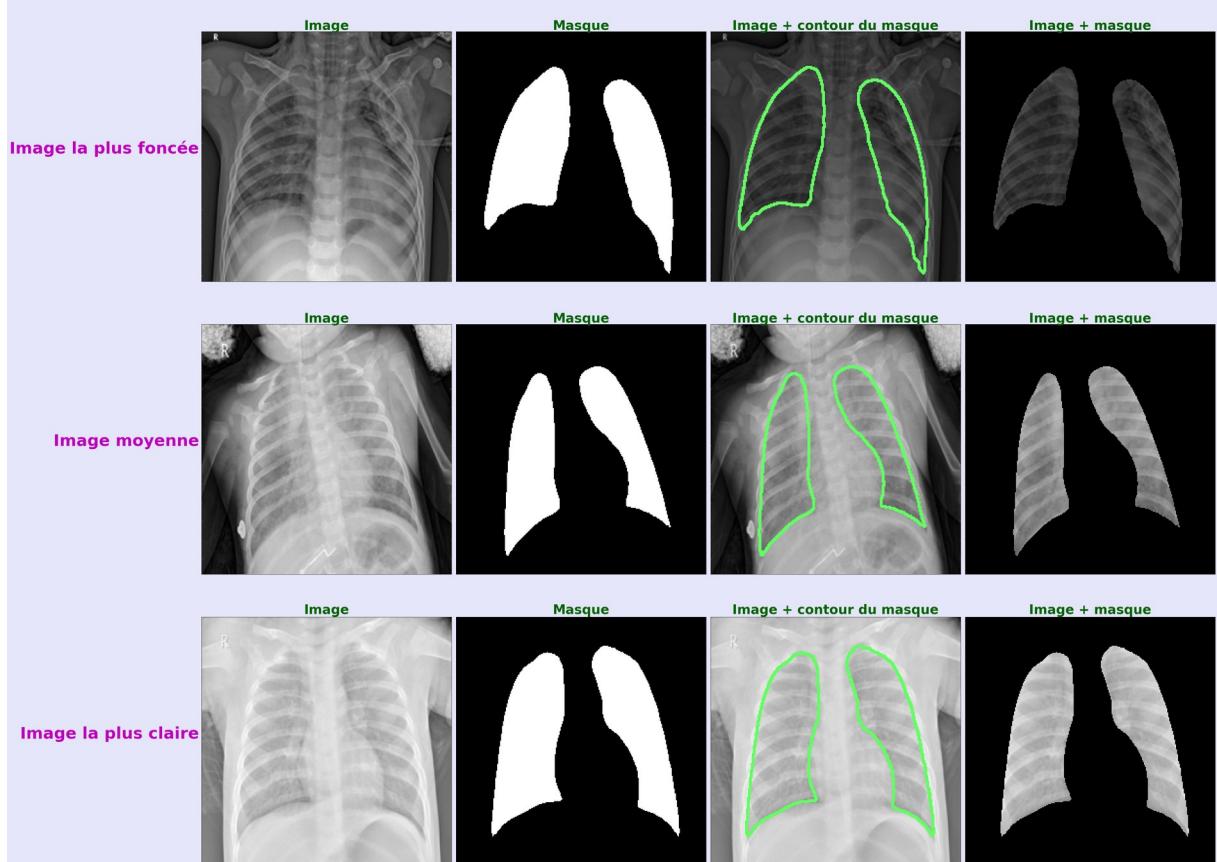
Nous affichons pour chaque catégorie l'image la plus foncée, d'intensité moyenne et la plus claire avec son masque, le contour du masque appliqué pour s'assurer que le masque correspond bien, ainsi que l'image avec le masque appliqué :



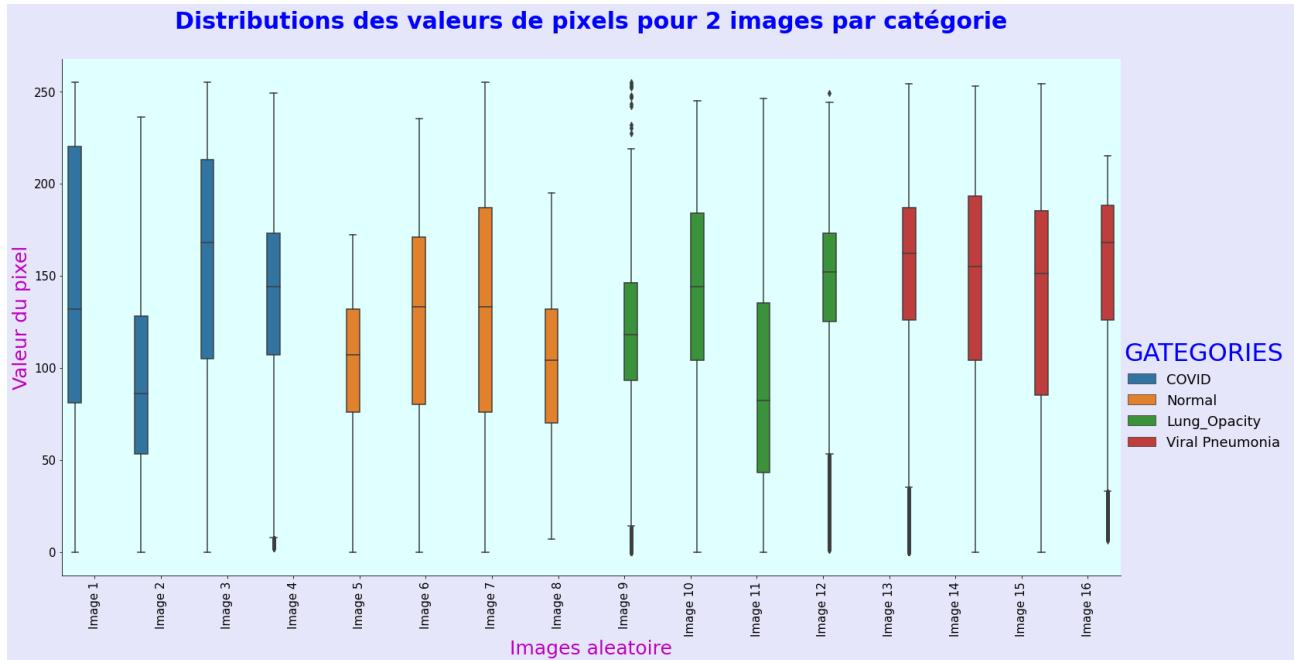
### Images Lung\_Opacity



### Images Viral Pneumonia



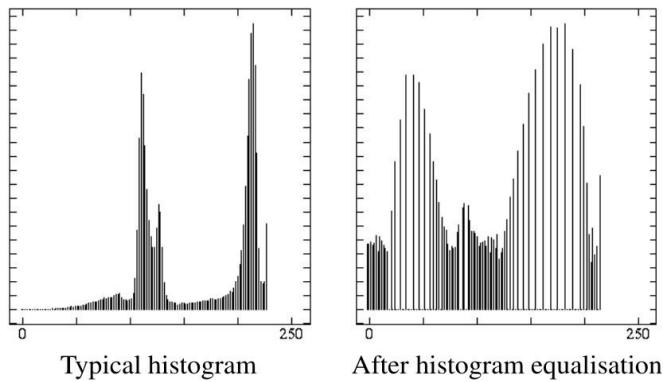
Nous nous sommes rapidement aperçus que les images manquaient de contraste et nous nous sommes donc intéressés à la luminosité générale des différentes radiographies:



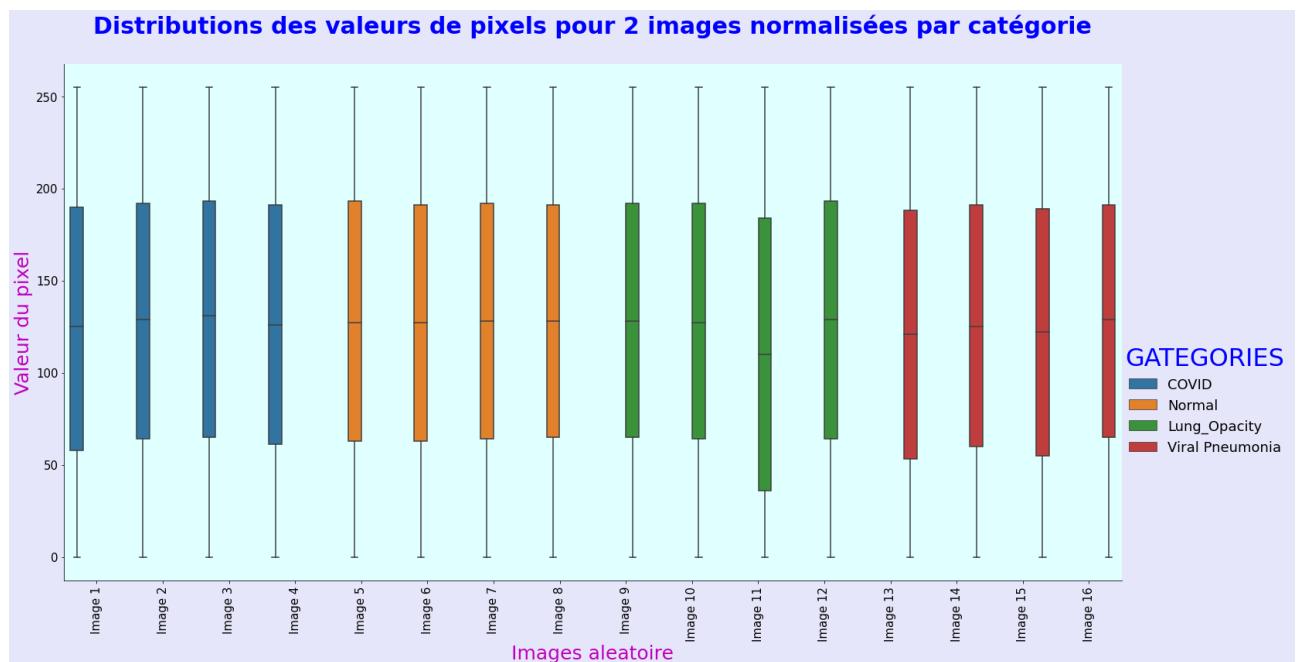
En affichant la distribution des pixels sur quelques images tirées de manière aléatoire, on remarque que la valeur médiane est différente suivant la catégorie de la radio et que les pixels ne sont pas repartis sur toute la plage (0, 255). Il va donc être nécessaire de normaliser les images.

Nous avons choisi de réaliser une normalisation par histogramme. Cela consiste à considérer la représentation graphique des distributions d'intensité d'une image, et à l'étendre sur toute la plage de valeurs de 0 à 255. On peut alors quantifier le nombre de pixels pour chaque valeur d'intensité. A partir de l'histogramme de l'image, nous allons chercher à rendre la distribution d'intensité plus large et uniforme afin que les valeurs d'intensité soient mieux réparties. Ainsi, si beaucoup de pixels sont dans une gamme réduite d'intensité, la normalisation va répartir ces mêmes pixels sur une gamme d'intensité plus large, afin de représenter la même image avec une plus grande variété de niveaux de gris. Cela a pour effet d'améliorer le contraste et donc l'apparition de détails sur l'image finale. L'image ci-dessous permet de mieux comprendre la transformation de la répartitions des pixels.

## Histogram equalisation examples

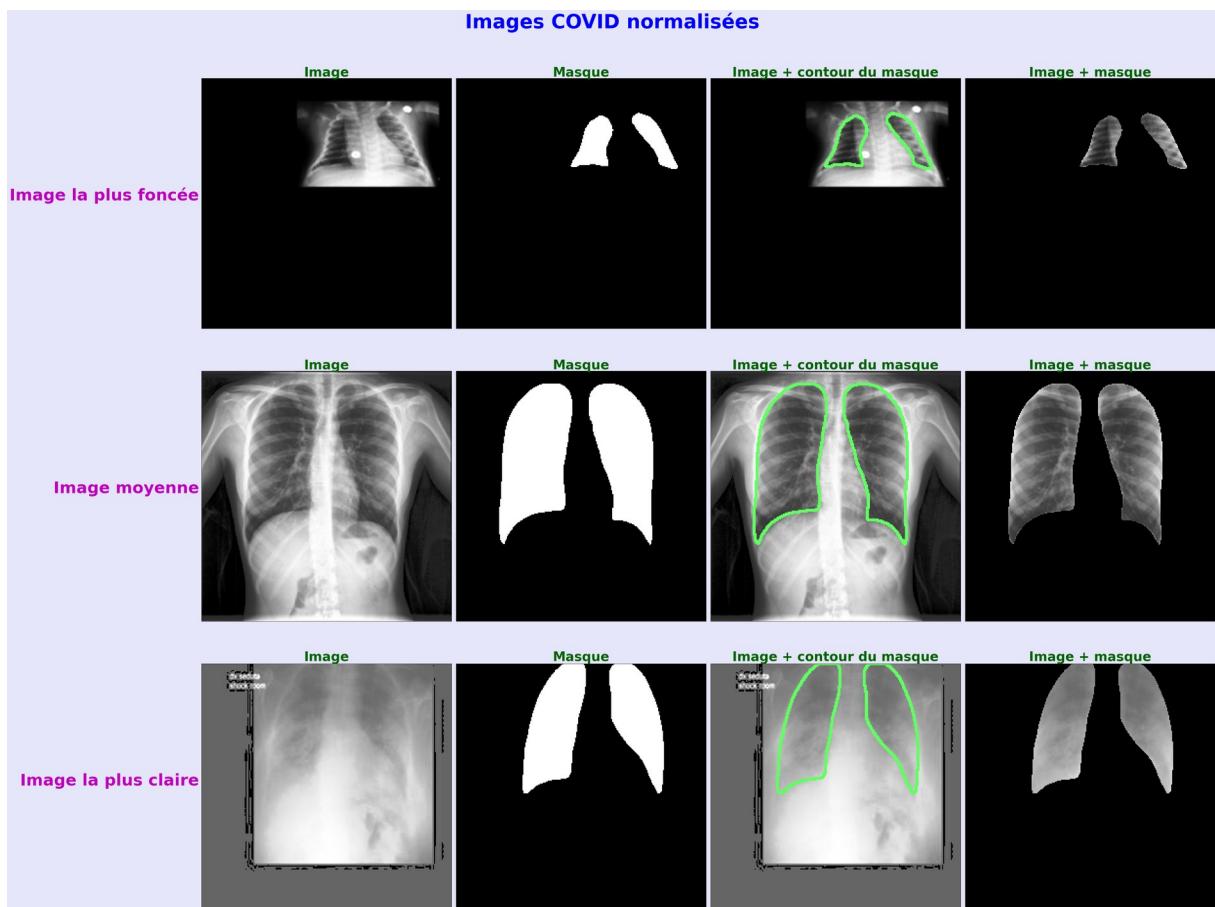
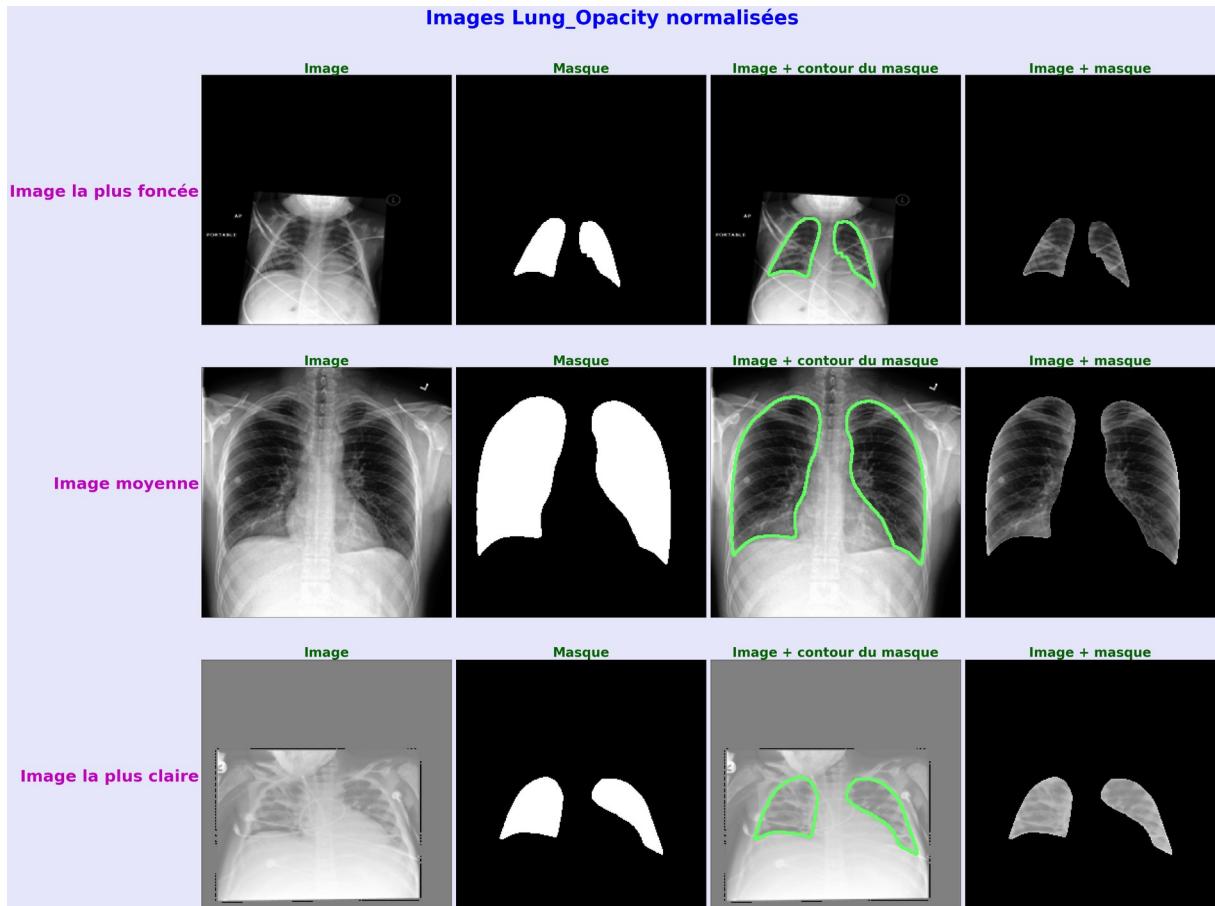


Nous avons donc équilibré toutes nos images et nous affichons de nouveau la distribution des pixels :

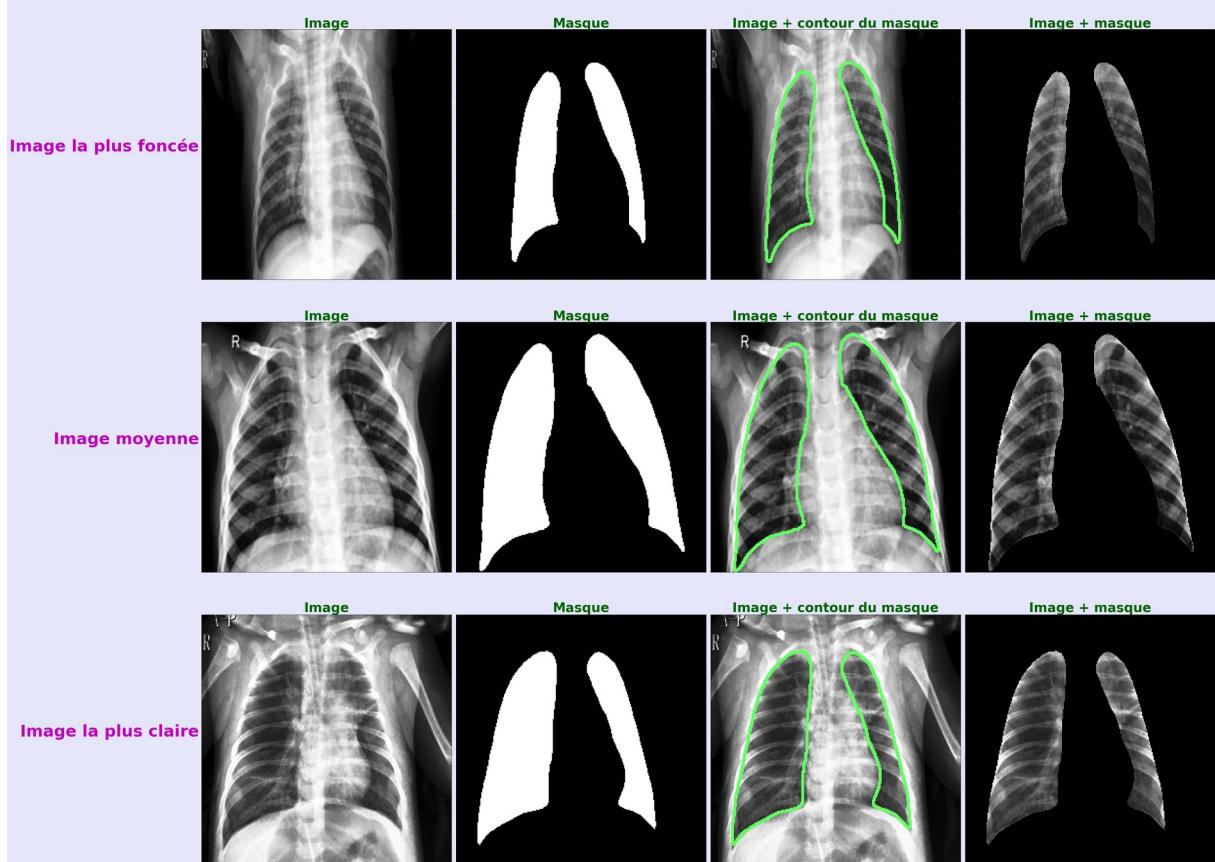


En affichant la distribution des pixels sur quelques images normalisées tirées de manière aléatoire, on remarque que les valeurs médianes sont plus proches et que les pixels sont bien repartis sur toute la plage (0, 255). Nous pensons aussi que par ce moyen, une répartition plus uniforme de la luminosité des pixels permettra d'éliminer toute forme de biais de décision pour le modèle de catégorisation final.

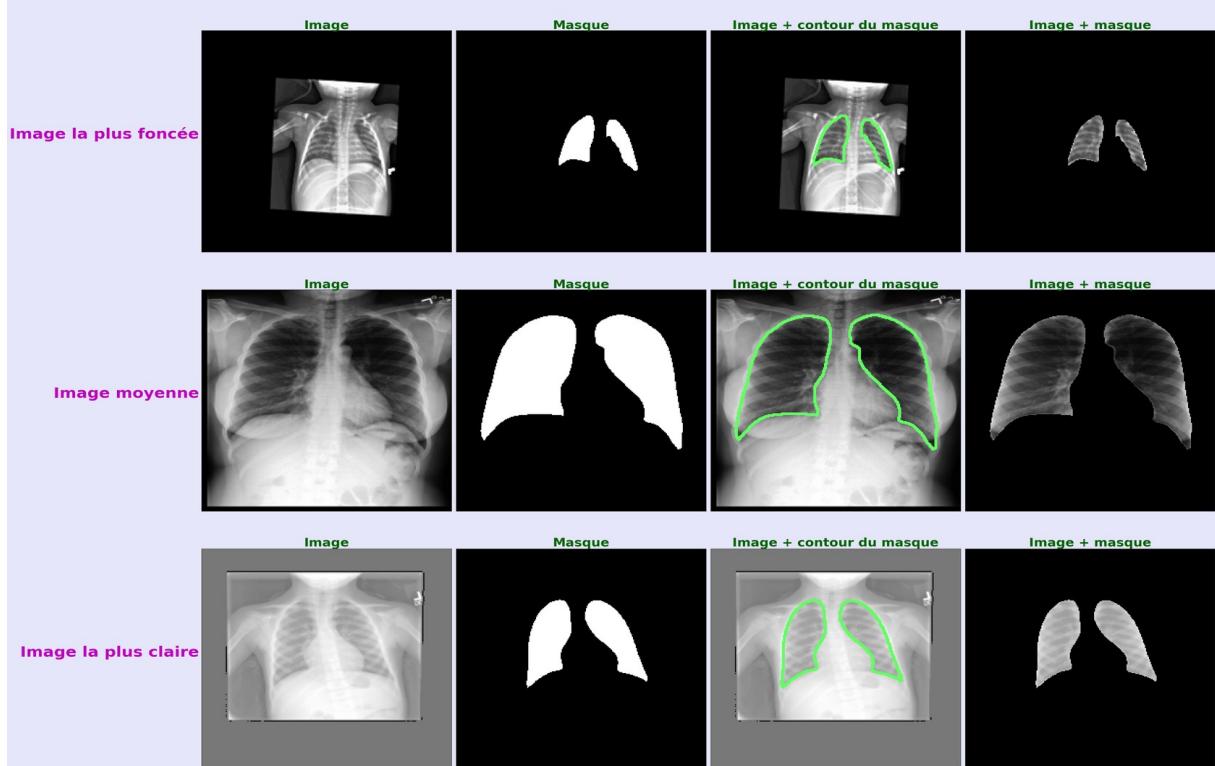
Nous affichons de nouveau pour chaque catégorie l'image la plus foncée, d'intensité moyenne et la plus claire avec son masque, le contour du masque appliqué pour s'assurer que le masque correspond bien, ainsi que l'image avec le masque appliqué



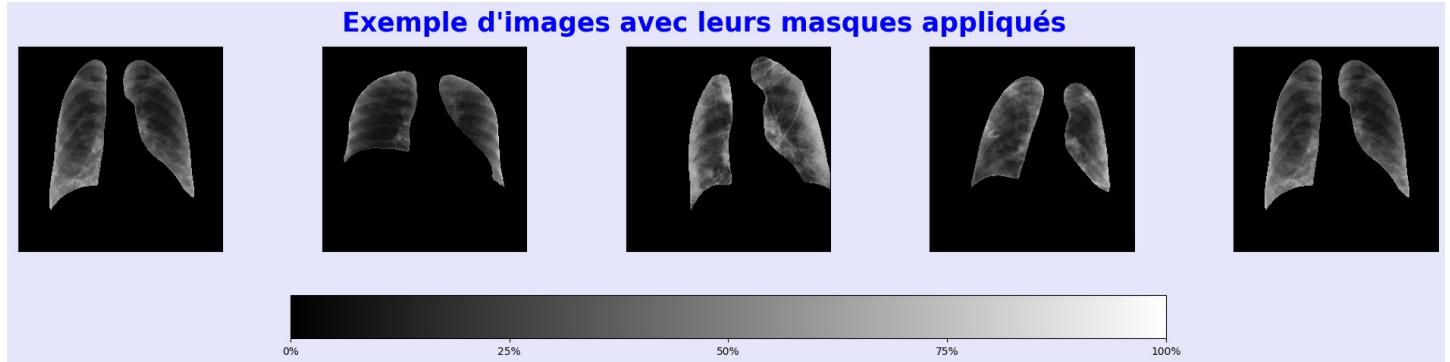
### Images Viral Pneumonia normalisées



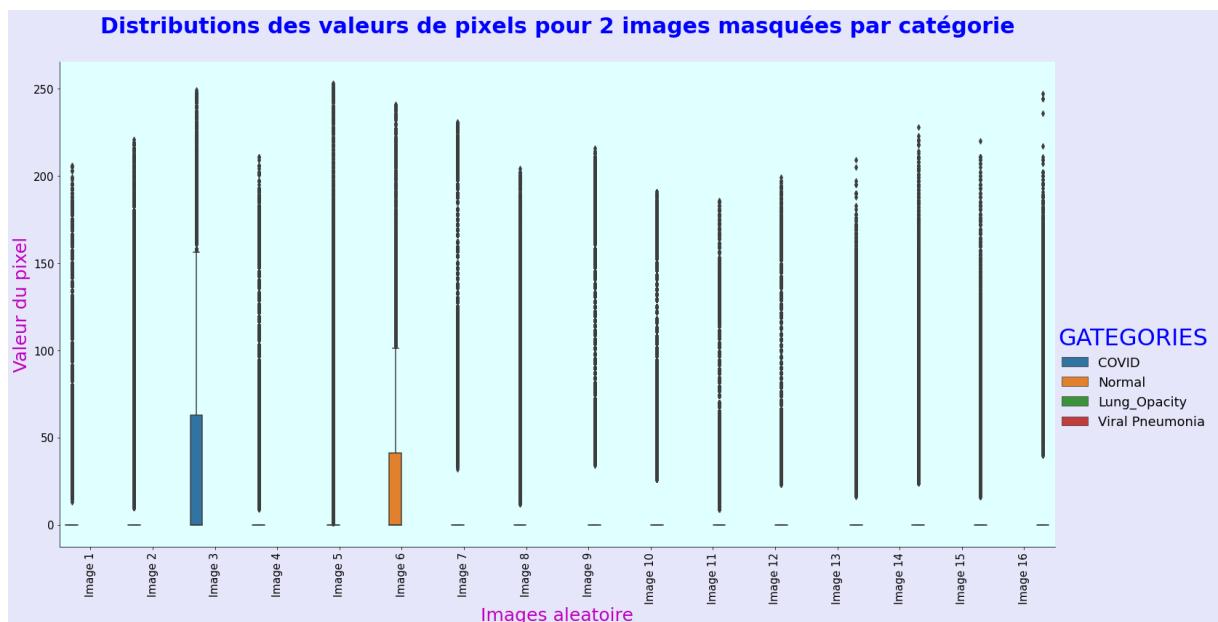
### Images Normal normalisées



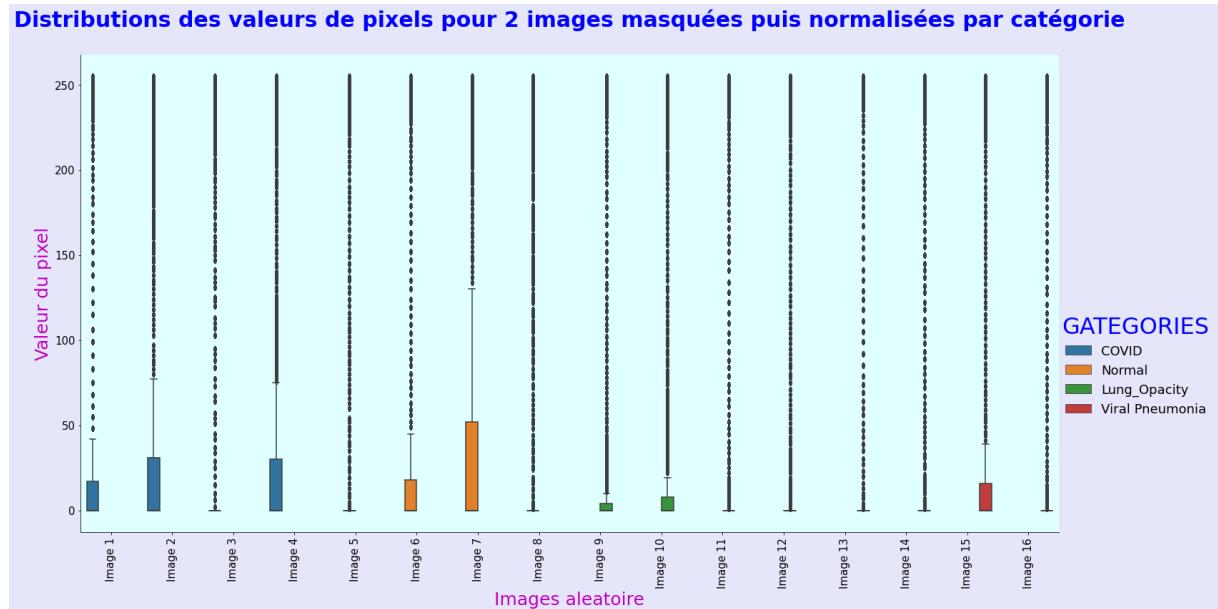
Nous constatons que le contraste est bien meilleur avec les images normalisées. Mais en comparant les images avec le masque on se rends compte qu'elle manque de contraste également :



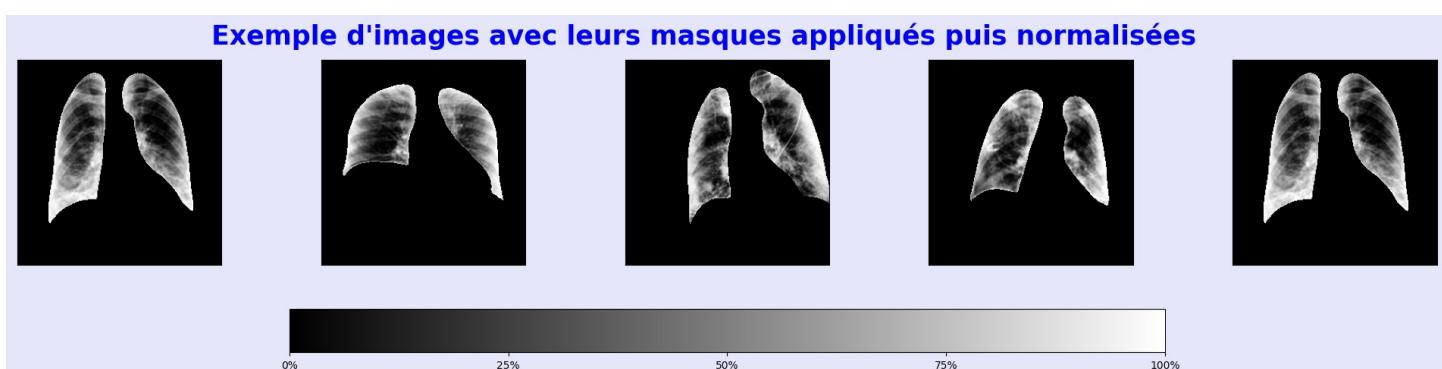
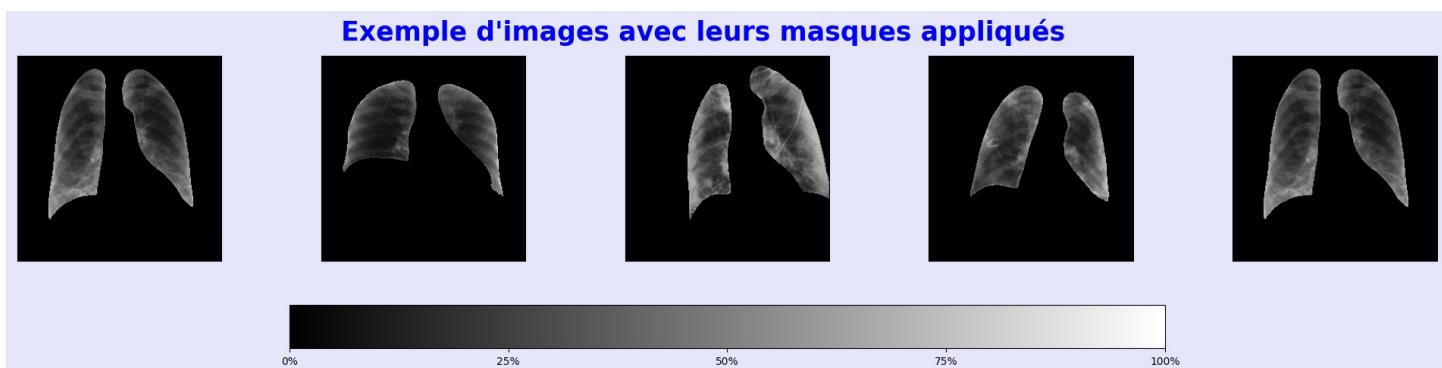
Nous avons aussi constaté qu'il faudrait aussi normaliser les images avec l'application du masque en affichant la distribution des pixels des images avec masque choisies aléatoirement :



On remarque que les pixels ne sont pas repartis sur toute la plage (0, 255). On normalise donc les images avec masque et on affiche de nouveau la distribution des pixels sur ces images avec masque normalisées :



Cette distribution nous permet de confirmer que la normalisation des images avec masque a bien été exécutée en comparant avec des images avec masque sans normalisation et avec normalisation :

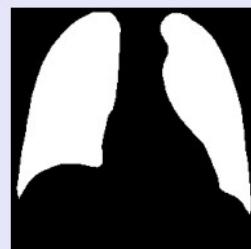


On remarque que le contraste après normalisation sur les images avec masque est bien meilleur. Affichons maintenant les images finales :

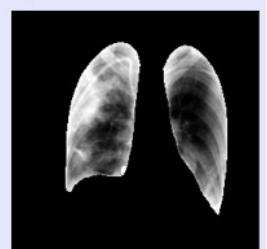
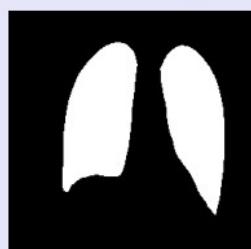
#### Exemple d'images de la catégorie: Normal



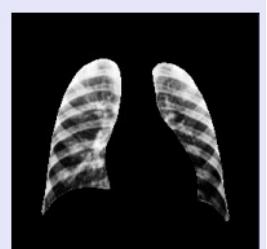
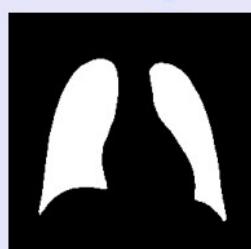
#### Exemple d'images de la catégorie: COVID



#### Exemple d'images de la catégorie: Lung\_Opacity



#### Exemple d'images de la catégorie: Viral Pneumonia

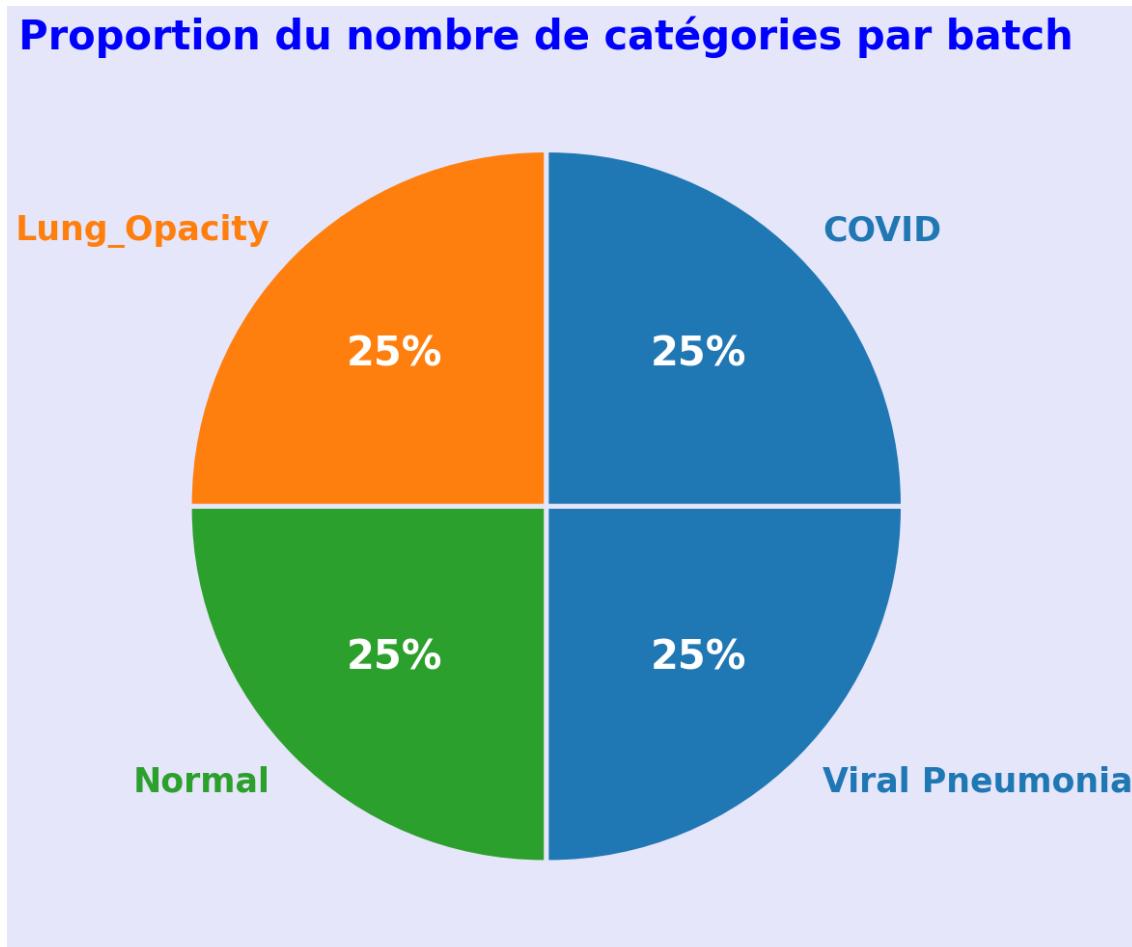


#### 4) Gestion des labels et de l'équilibre du dataset

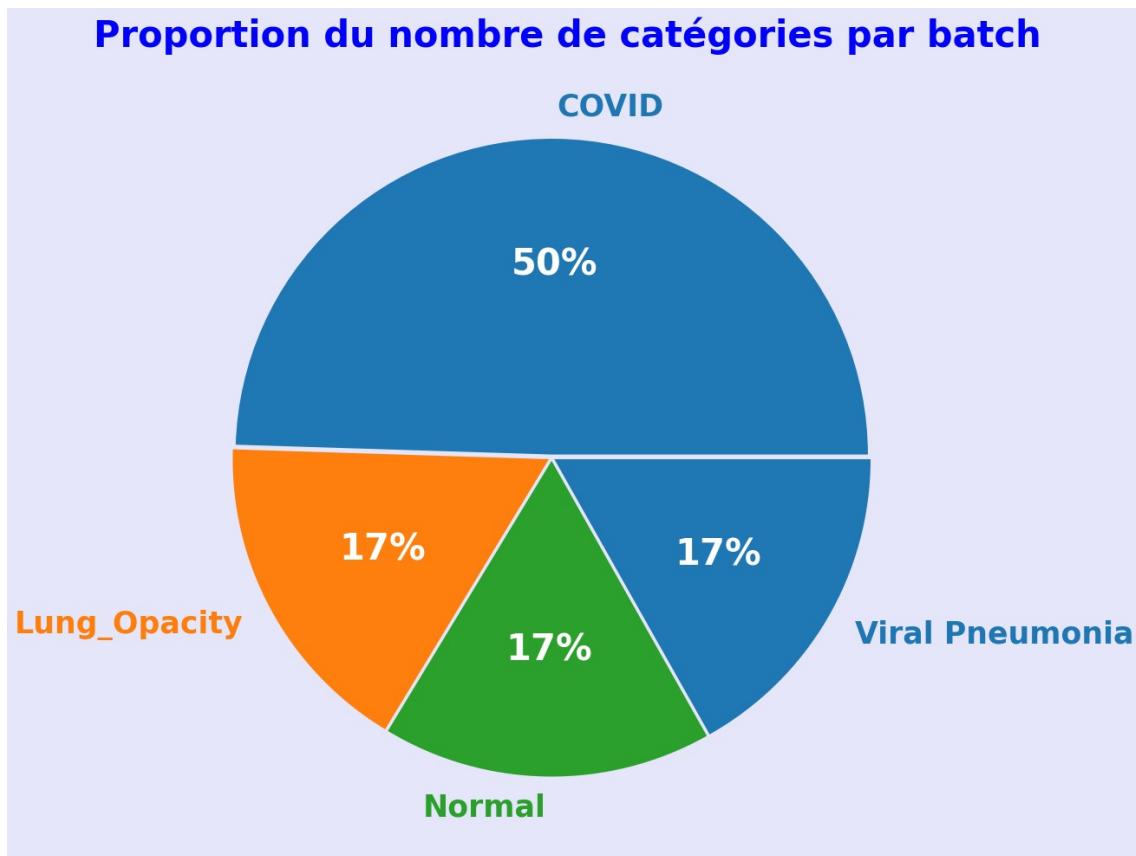
Le but du modèle de prédiction étant de détecter les cas de covid-19, la question suivante est donc posée : doit-on conserver les 4 labels précédents (covid, normal, opacity et pneumonie) ou bien réduire le nombre de labels à seulement deux (covid / non- covid) en regroupant les 3 dernières images dans la même catégorie ?

Nous avons exploré différentes possibilités, tout en tentant du mieux possible d'avoir un dataset plus équilibré d'abord à l'aide de la librairie imblearn et de ses fonctions RandomUnderSampler et RandomOverSampler. Les images étant de dimensions conséquentes (256 x 256) et les capacités de RAM étant limitées sur notebook, il nous était également important de réduire l'ensemble d'entraînement à une taille à la fois acceptable pour la mémoire que nous avons à disposition, et en même temps suffisante pour obtenir de bonnes performances pendant l'entraînement du modèle.

Une autre piste envisagée fut l'utilisation de l'API dataset de Tensorflow, pourvue de méthodes permettant d'équilibrer les batchs en termes de labels, mais aussi de pouvoir faire de l'augmentation de données, et ce d'une manière à la fois rapide, contrôlable et économique en terme de mémoire. Cette piste fut fructueuse pour la partie augmentation de données et gestion de la mémoire, mais insuffisante quand il a fallu équilibrer notre dataset. Nous sommes donc venus à créer notre propre générateur de batchs, de manière à contrôler l'équilibre des labels fournis au modèle à chaque step. L'idée initiale fut de faire apparaître plusieurs fois des exemples des classes minoritaires tirés aléatoirement, de manière à garantir que chaque batch (et donc chaque epoch) contienne 25 % de COVID, 25 % de Normal, 25 % de Lung Opacity et 25 % de Viral Pneumonia, tout en exposant la totalité du dataset au modèle.



Dans le cas d'une classification à 4 labels, ceci fut très fructueux et nous avons pu observer des gains de scores conséquents, mais ceci s'est révélé insuffisant dans le cas d'une classification binaire avec mutualisation des labels dans une classe antagoniste (Covid Vs all). Nous avons donc fait évoluer le générateur, de manière à garantir que, dans ce cas, chaque batch soit à nouveau équilibré en termes de classes, et que la classe mutualisée soit elle-même équitablement répartie entre les classes la composant. Par exemple, dans le cas d'une classification binaire Covid Vs All, chaque batch se composait de 50 % de Covid, et 50 % d'une classe « non-Covid », elle-même composée de 1/3 de Normal, 1/3 de Lung Opacity et 1/3 de Viral Pneumonia.



Cette méthode nous a permis à nouveau d'observer un gain de score conséquent. Nous avons donc validé cette méthode comme celle qui semblait idéale dans notre cas.

# Modélisation

## 2. Choix du modèle et motivations

Nous faisons face à un problème de classification sur données déséquilibrées. Nous disposons d'une forte disparité des classes à prédire. Il nous faut assigner une classe parmi les quatre présentées à chaque couple radio/masque. Comme notre jeu de donnée est déséquilibré, on peut comparer ce problème à une détection de fraude en considérant que les « fraudes » sont les classes Covid que l'on cherche à prédire. Nous avons choisi de nous concentrer sur la précision ( *accuracy* ) en tant que métrique pour superviser les scores sur les set d'entraînement et de validation, mais cette métrique est in fine insuffisante pour le cas d'un problème de détection de maladie : Il nous faut certes avoir un maximum de réponses correctes sur le nombres de cas de COVID détectés, mais il nous faut surtout détecter le plus grand nombre possible de cas parmi ceux présents dans le set.

Or comme nous l'avons vu, le jeu de donnée étant déséquilibré, nous pourrions obtenir un bon score de précision général pour le modèle sans pour autant que celui-ci ne prédise correctement la classe Covid. C'est généralement le problème dans les cas de détection de fraude. Afin d'éviter ce problème nous avions deux choix à notre disposition :

- Choisir une métrique supplémentaire tels que le rappel afin de savoir si la classe Covid avait été correctement détecter par l'algorithme.
- Equilibrer le jeu de donnée lors de l'entraînement du modèle afin de contourner le problème.

Nous avons choisi de créer un jeu de donnée équilibré, tout en nous assurant d'obtenir des résultats satisfaisants sur les métriques de précision et de rappel.

Nous avons convenu du pipeline suivant pour la classification d'une radio non masquée :

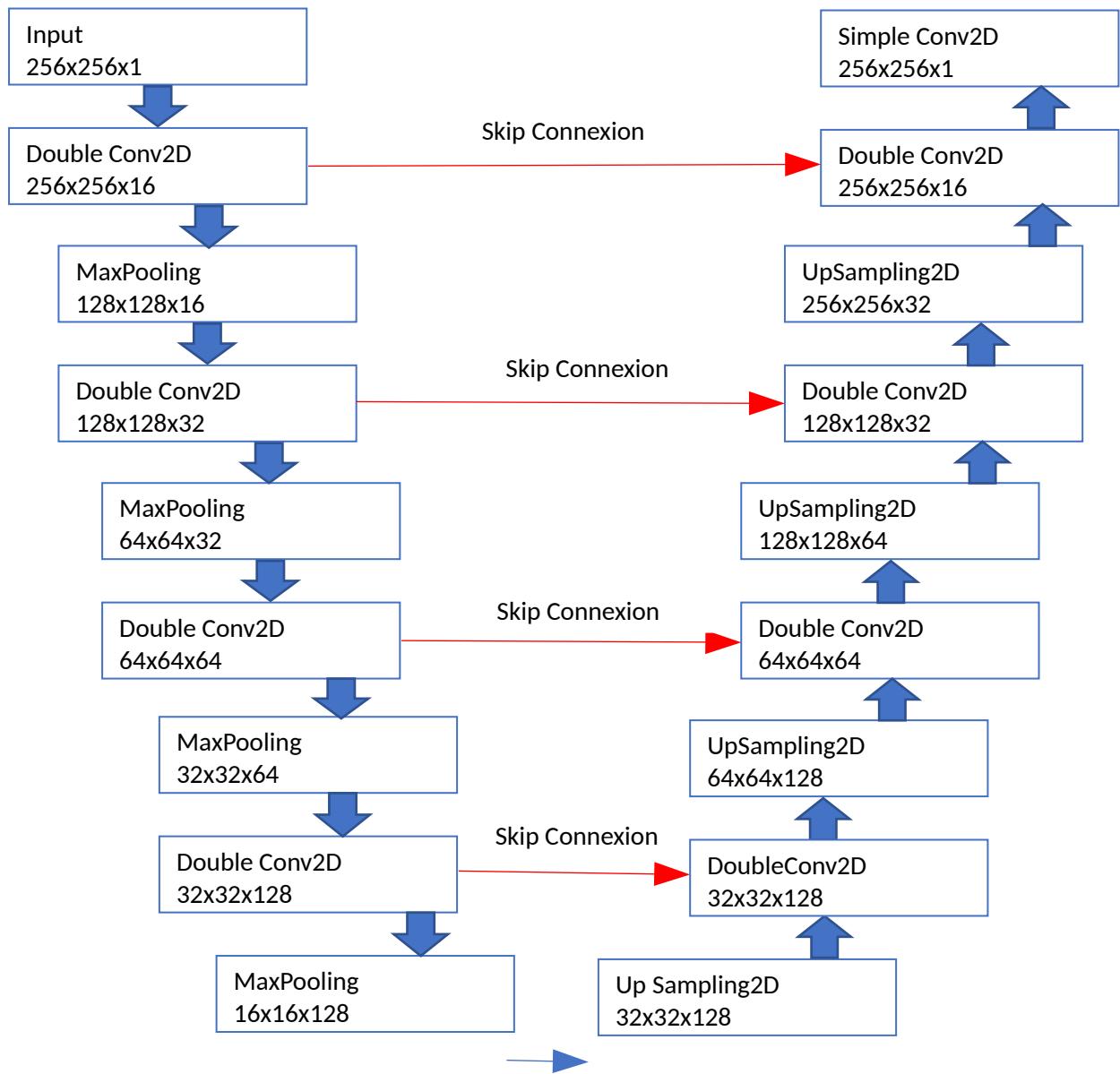
- pre-processing des données (rescaling en 256x256 puis égalisation par histogramme via la librairie cv2)
- utilisation d'un modèle de segmentation de type U-net afin de déterminer le masque des poumons correspondant à une radiographie
- application du masque calculé par le modèle précédent sur la radio afin de créer une nouvelle image composée uniquement des poumons, et à nouveau égalisation par histogramme sur l'image résultante
- Utilisation d'un CNN de classification
- Visualisation des zones d'intérêts du modèle à l'aide d'un modèle GradCam

## 2. Segmentation d'images

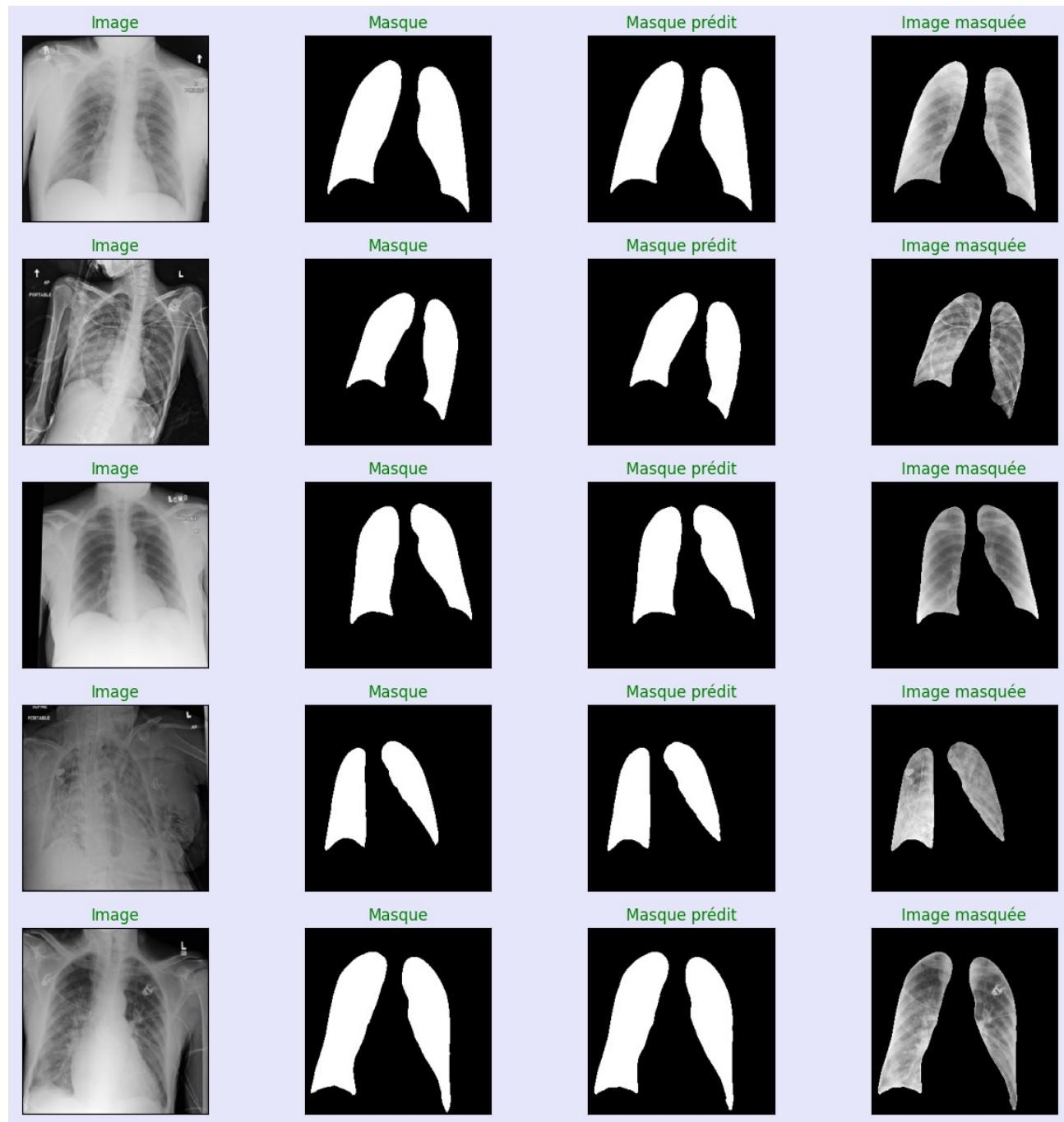
Pour la partie segmentation, les labels n'étant pas utiles pour déterminer le masque des poumons, nous avons utilisé la fonction méthode `generator.flow_from_directory` afin de charger les images de radiographie d'un côté, et les images de masques d'un autre, puis nous avons zippé ces 2 generators, afin d'utiliser le premier comme les features 'entrées, et le second comme les targets. La fonction de perte alors utilisée est `Binary Crossentropy` et la dernière couche est une `conv2D à 1 filtre unique`, avec activation un `sigmoid`.

Le modèle utilisé est un U-net, utilisant pour la partie encoder 5 blocs `downsampling` composé de 2 couches de convolution 2D identiques suivies d'un `maxpooling`, avec un nombre de filtres allant de 16 à 128, activation `relu`. La partie `decoder` est quant à elle constituée de 5 blocs composés d'une couche `UpSampling2D`, puis de 2 couches de convolution 2D identiques, avec un nombre de filtres allant de 128 à 16. La couche de sortie est une simple convolution 2D équipée de 1 filtre d'activation `sigmoid`, et des `skip connexion` sont effectuées entre chaque couche de dimensions identiques de l'encoder au decoder. Une fois la prédiction effectuée, nous arrondissons à l'entier le plus proche (soit 0 ou 1, du fait de la normalisation).

Représentation graphique du model U-net :



Après construction du modèle, nous avons entraîné ce modèle sur 90 % du dataset, et sommes arrivés à une accuracy finale de 99.2 % par rapport au set de validation. Notre modèle découpe donc parfaitement les poumons dans les images initiales et permet de générer un masque sous forme de matrice 256 x 256 composée de 0 et de 1.



### 3. Classification des images masquées

#### II. Architecture CNN « from scratch »

##### 1<sup>er</sup> modèle CNN

Dans un premier temps nous avons entraîné notre modèle sur trois classes différentes : 0 = Covid, 1 = Normal, 2 = Pneumonia + Lung Opacity

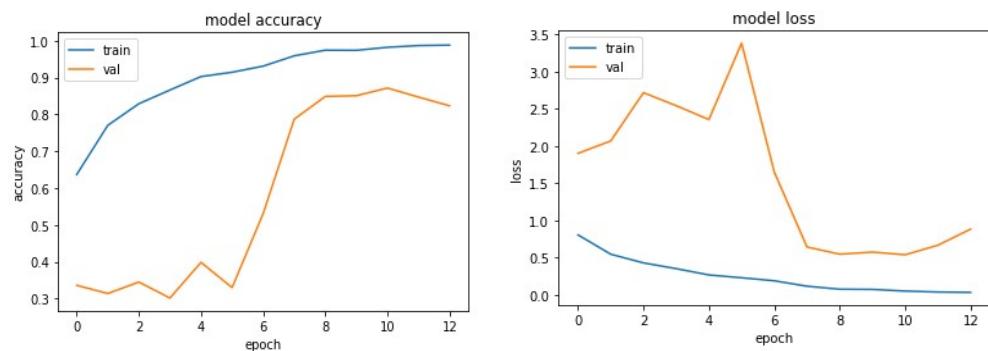
Nous avons donc regroupé les classes Pneumonie Aiguë et Opacité Pulmonaire sous un même label. Nous avons ensuite utilisé un RandomUnderSampler afin d'équilibrer le jeu de donnée. Le summary du modèle utilisé est le suivant :

```
Model: "model"

-----  
Layer (type)          Output Shape       Param #  
-----  
input_1 (InputLayer)    [(None, 256, 256)]     0  
-----  
reshape (Reshape)      (None, 256, 256, 1)     0  
-----  
batch_normalization (BatchNo (None, 256, 256, 1) 4  
-----  
conv2d (Conv2D)         (None, 256, 256, 16)    1824  
-----  
max_pooling2d (MaxPooling2D) (None, 64, 64, 16)    0  
-----  
batch_normalization_1 (Batch (None, 64, 64, 16)    64  
-----  
conv2d_1 (Conv2D)        (None, 64, 64, 32)    32768  
-----  
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)    0  
-----  
batch_normalization_2 (Batch (None, 16, 16, 32)    128  
-----  
conv2d_2 (Conv2D)        (None, 16, 16, 64)    131072  
-----  
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)    0  
-----  
batch_normalization_3 (Batch (None, 4, 4, 64)    256  
-----  
flatten (Flatten)        (None, 1024)        0  
-----  
dense (Dense)            (None, 256)        262400  
-----  
dense_1 (Dense)          (None, 64)        16448  
-----  
dense_2 (Dense)          (None, 3)        195  
-----  
Total params: 444,359  
Trainable params: 444,133  
Non-trainable params: 226  
-----
```

On entraîne le modèle sur 73 epochs, avec un optimiseur « Adam » (lr =0.001), une loss « CategoricalCrossentropy » (dernière activation softmax) et un batch size de 128.

On peut voir ci-dessous pour respectivement, l'entraînement et la validation, les courbes des scores d'accuracy et des valeurs des fonctions de pertes.



Une fois l'entraînement terminé, nous avons effectué des prédictions à partir du set de test et avons projeté la matrice de confusion :

	precision	recall	f1-score	support
0	0.78	0.72	0.75	373
1	0.89	0.93	0.91	1033
2	0.89	0.87	0.88	711
accuracy			0.87	2117
macro avg		0.85	0.84	2117
weighted avg		0.87	0.87	2117

col_0	0	1	2
row_0			
0	270	62	41
1	36	959	38
2	38	57	616

On peut constater grâce à ce tableau que la classe COVID est celle qui se retrouve la plus confondue avec les 2 autres classes. Tant sur le recall que sur sa précision, ce label semble difficile à distinguer des autres.

## 2nd Modèle CNN

Nous avons ensuite décidé de réduire le nombre de labels de 3 à 2 : Covid ou non covid (c'est à dire normal, pneumonia ou lung-opacity). Afin de compenser les écarts cardinaux dans les 2 ensembles résultants (environ 3000 covid contre 18000 non-covid), nous avons fait les choix suivants :

- effectuer un over-sampling aléatoire des données sur la classe minoritaire dans l'échantillon d'entraînement
- effectuer un under-sampling aléatoire des données sur la classe majoritaire dans les échantillons de validation et de test
- utiliser un ImageDataGenerator de la librairie tf.keras afin de modifier les images des deux classes, en apportant un zoom aléatoire de  $\pm 10\%$ , un height/width shift de  $\pm 10\%$ , une symétrie horizontale aléatoire et une symétrie verticale aléatoire.

- Placer les sets de validation et d'entraînement dans des datasets tensorflow afin de garantir une meilleure utilisation des performances et de la mémoire RAM.

Nous nous sommes par la suite aperçus que l'utilisation des méthodes d'un générateur pour faire de l'augmentation d'images n'était pas prise en charge par le GPU. Nous avons donc décidé par la suite d'effectuer les modifications aléatoires directement au sein du modèle, grâce aux layers RandomFlip, RandomZoom et RandomRotation, qui elles sont prises en charge par le GPU. Ces layers ne s'exécutent que pendant l'entraînement, mais pas pour la validation ou la prédiction. On peut voir ci-dessous le summary() du modèle :

```
Model: "model"
-----  

Layer (type)          Output Shape         Param #  

-----  

input_1 (InputLayer)   [(None, 256, 256, 1)]  0  

-----  

random_flip (RandomFlip)  (None, 256, 256, 1)  0  

-----  

random_rotation (RandomRotation) (None, 256, 256, 1)  0  

-----  

random_zoom (RandomZoom)  (None, 256, 256, 1)  0  

-----  

rescaling (Rescaling)    (None, 256, 256, 1)  0  

-----  

conv2d (Conv2D)         (None, 249, 249, 16)   1024  

-----  

max_pooling2d (MaxPooling2D) (None, 124, 124, 16)  0  

-----  

conv2d_1 (Conv2D)       (None, 117, 117, 32)   32768  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 58, 58, 32)  0  

-----  

conv2d_2 (Conv2D)       (None, 51, 51, 64)    131072  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 25, 25, 64)  0  

-----  

conv2d_3 (Conv2D)       (None, 18, 18, 128)   524288  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 9, 9, 128)  0  

-----  

conv2d_4 (Conv2D)       (None, 2, 2, 256)    2097152  

-----  

flatten (Flatten)       (None, 1024)        0  

-----  

dense (Dense)          (None, 256)        262400  

-----  

dense_1 (Dense)        (None, 64)        16448  

-----  

dense_2 (Dense)        (None, 1)        65  

-----  

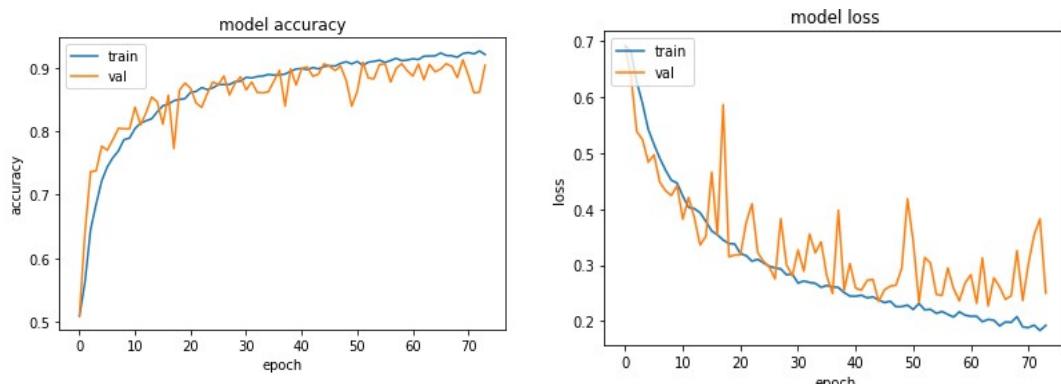
Total params: 3,065,217  

Trainable params: 3,065,217  

Non-trainable params: 0
```

On entraîne le modèle sur 73 epochs, avec un optimiseur « Adam » (lr =0.001), une loss « BinaryCrossentropy » (dernière activation sigmoid) et un batch size de 128.

Et ci-dessous, pour l'entraînement et la validation, on peut voir, respectivement, les courbes des scores d'accuracy et des valeurs de la fonction de perte.



Une fois l'entraînement effectué, nous avons effectué des prédictions à partir du set de test et avons projeté la matrice de confusion.

	precision	recall	f1-score	support
0	0.89	0.83	0.86	373
1	0.84	0.90	0.87	373
accuracy			0.87	746
macro avg	0.87	0.87	0.87	746
weighted avg	0.87	0.87	0.87	746

col_0	0	1
row_0		
0	311	62
1	38	335

Les résultats sont corrects, mais on constate que quelques cas de COVID sont mal classés.

### Modèle CNN final

Afin de palier le déséquilibre des classes sans toutefois perdre de données, et nous assurer d'une répartition équitable des données au sein de chaque batch, nous avons décidé de développer notre propre générateur (voir partie 4 du chapitre Manipulation des données). Celui-ci permet, initialement, de répartir équitablement les 4 classes au sein de chaque batch, en faisant apparaître plusieurs fois à chaque epoch des exemples des classes minoritaires au sein d'autres batchs. Nous l'avons par la suite amélioré de manière à ce que, dans le cas d'une mutualisation des classes (comme dans la classification binaire ici effectuée), les classes restantes soient encore équilibrées dans les batch, tout en garantissant une représentation équilibrée des différents labels au sein de la classe mutualisée.

Nous avons également décidé de remplacer les couches de convolution 2D par des couches de Convolution 2D séparables, similaires à celles présentes dans le modèle MobileNet. Ces couches utilisent un système de multiplication « pointwise » qui permet de réduire le coût des convolutions par kernel classique ont pour particularité d'être extrêmement légères, et permettent également d'atteindre des résultats parfois supérieurs aux convolution classiques. Nous avons donc décidé de nous rapprocher d'une architecture plus grande : Nous avons doublé chaque couche de convolution pour chaque type de dimension et avons introduit une normalisation de batch, pour des dimensions allant de 16 à 128 par couche, et avons fixé le paramètre « pointwise depth » à 2.

Nous avons également remplacé la couche Flatten() par une couche de GlobalMaxPooling2D, à la fois plus économique et garantissant une extraction des couches précédentes plus pertinente.

Enfin, dans un but de nous rapprocher d'un modèle final complet, nous avons raccordé le modèle Unet en amont du CNN, et inclus une couche de préprocessing custom (afin d'effectuer la fonction d'égalisation par histogramme de cv2 au sein des couches du modèle et ainsi profiter de la puissance de calcul du GPU). Voici le résumé du modèle :

Model: "model_1"			
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 299, 299, 1 0 )]		[]
resizing (Resizing)	(None, 256, 256, 1) 0		['input_1[0][0]']
rescaling (Rescaling)	(None, 256, 256, 1) 0		['resizing[0][0]']
model (Functional)	(None, 256, 256, 1) 599073		['rescaling[0][0]']
tf.math.round (TF0pLambda)	(None, 256, 256, 1) 0		['model[0][0]']
mask (Multiply)	(None, 256, 256, 1) 0		['rescaling[0][0]', 'tf.math.round[0][0]']

Ci-dessus, l'input du modèle, les couches de resizing et rescaling ainsi que le raccordement du Unet pour le calcul du masque. Le masquage de l'image est obtenu par simple multiplication element-wise.

hist_norm (HistNorm)	(None, 256, 256, 1) 0	['mask[0][0]']
random_flip (RandomFlip)	(None, 256, 256, 1) 0	['hist_norm[0][0]']
random_rotation (RandomRotatio n)	(None, 256, 256, 1) 0	['random_flip[0][0]']

Ici, la suite de la partie pré-processing et l'augmentation des données par symétrie verticale aléatoire et rotation aléatoire. On remarque l'absence de RandomZoom, car nous avons constaté que cette augmentation ne semblait pas forcément utile pour réduire l'overfitting.

separable_conv2d (SeparableConv2D)	(None, 256, 256, 16 64)	['random_rotation[0][0]']
activation (Activation)	(None, 256, 256, 16 0)	['separable_conv2d[0][0]']
batch_normalization (BatchNormalization)	(None, 256, 256, 16 64)	['activation[0][0]']
separable_conv2d_1 (SeparableConv2D)	(None, 253, 253, 16 1024)	['batch_normalization[0][0]']
activation_1 (Activation)	(None, 253, 253, 16 0)	['separable_conv2d_1[0][0]']
max_pooling2d (MaxPooling2D)	(None, 127, 127, 16 0)	['activation_1[0][0]']
separable_conv2d_2 (SeparableConv2D)	(None, 127, 127, 32 1536)	['max_pooling2d[0][0]']
activation_2 (Activation)	(None, 127, 127, 32 0)	['separable_conv2d_2[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 127, 127, 32 128)	['activation_2[0][0]']
separable_conv2d_3 (SeparableConv2D)	(None, 124, 124, 32 3072)	['batch_normalization_1[0][0]']
activation_3 (Activation)	(None, 124, 124, 32 0)	['separable_conv2d_3[0][0]']

Ci-dessus, les 2 premières double couches de convolution 2D séparables. On notera la présence d'une couche batch-normalization entre chaque SeparableConv2D de dimension égale. Chaque « étage » de convolution a pour dimension le double de la dimension des couches de l'étage précédent. De plus, le paramètre « pointwise depth » est fixé à 2. Pour le MaxPooling, nous avons sélectionné un pool de taille (2, 2) et un stride de (1, 1).

```

separable_conv2d_4 (SeparableC  (None, 62, 62, 64)  5120      ['max_pooling2d_1[0][0]']

activation_4 (Activation)      (None, 62, 62, 64)  0      ['separable_conv2d_4[0][0]']

batch_normalization_2 (BatchNo  (None, 62, 62, 64)  256      ['activation_4[0][0]']

rmalization)

separable_conv2d_5 (SeparableC  (None, 59, 59, 64)  10240     ['batch_normalization_2[0][0]']

onv2D)

activation_5 (Activation)      (None, 59, 59, 64)  0      ['separable_conv2d_5[0][0]']

max_pooling2d_2 (MaxPooling2D) (None, 30, 30, 64)  0      ['activation_5[0][0]']

separable_conv2d_6 (SeparableC  (None, 30, 30, 128) 18432     ['max_pooling2d_2[0][0]']

onv2D)

activation_6 (Activation)      (None, 30, 30, 128)  0      ['separable_conv2d_6[0][0]']

batch_normalization_3 (BatchNo  (None, 30, 30, 128)  512      ['activation_6[0][0]']

rmalization)

separable_conv2d_7 (SeparableC  (None, 27, 27, 128) 36864     ['batch_normalization_3[0][0]']

onv2D)

activation_7 (Activation)      (None, 27, 27, 128)  0      ['separable_conv2d_7[0][0]']

global_max_pooling2d (GlobalMa  (None, 128)          0      ['activation_7[0][0]']

xPooling2D)

dense (Dense)                  (None, 16)           2064     ['global_max_pooling2d[0][0]']

dense_1 (Dense)                (None, 2)            34      ['dense[0][0]']

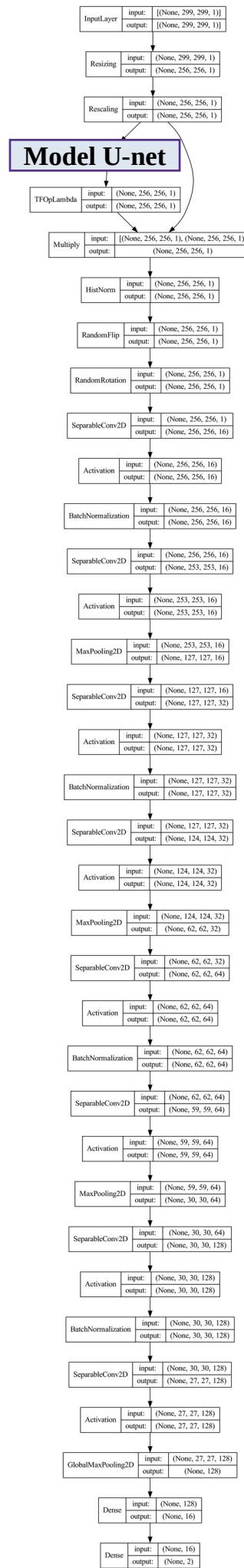
=====

Total params: 678,483
Trainable params: 78,930
Non-trainable params: 599,553

```

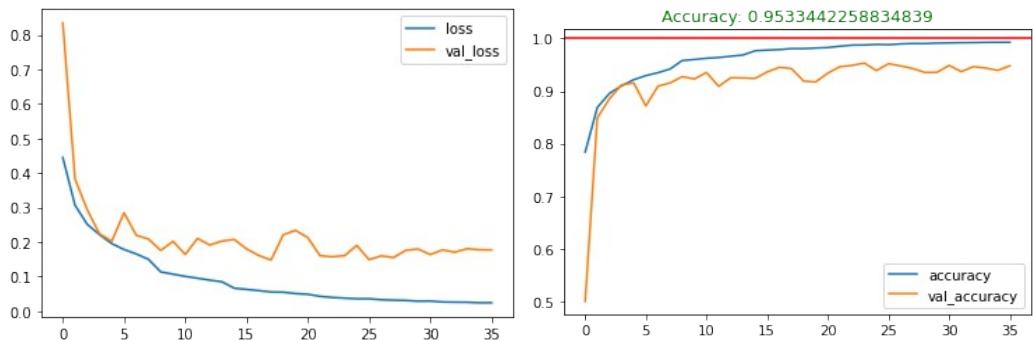
Enfin, on peut voir ici les 2 derniers étages de convolution 2D, la couche de GlobalMaxPooling2D et la partie classifier composée de 2 simples couches dense de 16 et 2 neurones chacune. L'activation finale est bien entendu un softmax. On notera le nombre particulièrement faible de paramètres pour composer ce modèle comparé aux deux précédents.

On peut voir sur la page suivante le schéma du modèle.



# Model 1

Le modèle est compilé avec un optimiseur Adam ( $lr = 0.001$ ), un batch de taille 128 et avec une loss « categorical crossentropy », comme pour les modèles précédents. Nous ajoutons également un callback de type ReduceLROnPlateau, afin de réduire le learning rate quand la valeur de la fonction de perte sur la partie validation ne baisse pendant plus de 3 epochs. Nous divisons alors le learning rate par 2, afin de poursuivre l'entraînement et obtenir les meilleures performances possibles, tout en évitant l'overfitting. Notre callback d'early-stopping cesse l'entraînement au bout de 26 epochs. Une fois l'entraînement terminé, nous affichons les courbes de la fonction de perte et de l'accuracy, pour le set d'entraînement et le set de validation :



Comme nous le voyons sur le graphique ci-dessus, nous obtenons alors le score de 95 % d'accuracy sur le set de validation. Nous vérifions alors ce résultat sur le set de test en dressant la matrice de confusion :

```
Confusion matrix:
[[169 11]
 [ 37 839]]
      precision    recall  f1-score   support
 0       0.82      0.94      0.88      180
 1       0.99      0.96      0.97      876

    accuracy          0.95      1056
   macro avg       0.90      0.95      0.92      1056
weighted avg       0.96      0.95      0.96      1056
```

Les résultats sont donc confirmés : le modèle est bien capable d'atteindre un F1 score global de 95 % sur la tâche de classification Covid Vs All. En adaptant le seuil de décision, qui est ici fixé par défaut à 0.5, nous obtenons des résultats parfois même meilleurs. Par exemple, pour un seuil minimum de 0.77 pour la classe Covid, nous obtenons la matrice de confusion suivante :

```
Confusion matrix:
[[164 16]
 [ 17 859]]
      precision    recall  f1-score   support
 0       0.91      0.91      0.91      180
 1       0.98      0.98      0.98      876

    accuracy          0.97      1056
   macro avg       0.94      0.95      0.94      1056
weighted avg       0.97      0.97      0.97      1056
```

Nous pouvons donc monter le modèle jusqu'à un score de 0.97 en F1 score, mais nous ne savons pas s'il s'agit d'un hasard dû à la répartition du set de test. Afin de vérifier cela, nous répétons la même opération avec un seuil de 0.7 sur le set de validation, et dressons la matrice de confusion :

Confusion matrix:				
		precision	recall	f1-score
		0	0.84	0.92
		1	0.98	0.96
				0.88
				180
				876
		accuracy		0.96
		macro avg	0.91	0.94
		weighted avg	0.96	0.96
				1056
				1056

Si nous n'atteignons pas les 0.97, nous constatons tout de même un résultat encore meilleur que celui affiché à la fin de l'entraînement. Nous pouvons en déduire que ce modèle est d'autant plus performant si son seuil de décision est légèrement relevé, entre 0.7 et 0.8.

Avec 94 % de recall sur le label Covid, nous pouvons au moins nous assurer du fait que notre modèle « repère » plutôt bien les cas de Covid. Sa précision peut être adaptée en réglant le seuil de décision.

## II. Fine-tuning d'un modèle pré-entraîné

Après avoir testé de nombreuses architectures différentes, nous avons décidé d'utiliser un modèle Xception pré-entraîné sur 'Imagenet', tout en gardant un modèle U-net en entrée.

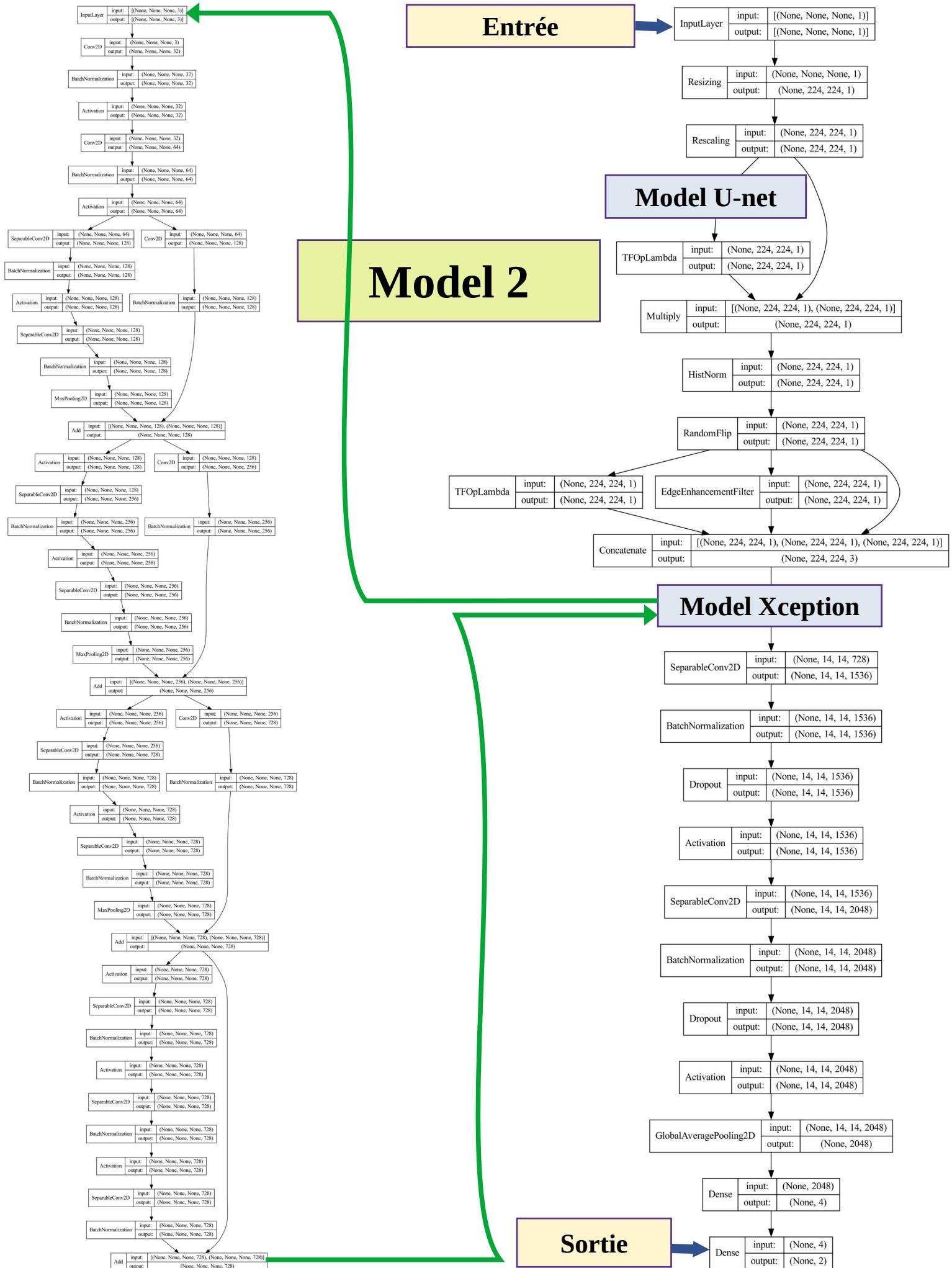
Notons que nous avons choisi de prendre en entrée une image de n'importe quelle taille et de faire tout le pré-processing au niveau des premières couches du modèle. Pour des raisons de compatibilité avec le modèle Xception, nous avons dû ré-entraîner notre modèle U-net avec des images de taille 224x224. Et avons fait le choix de rester sur 1 seul canal.

Étant donné que ce type d'images supporte mal les transformations pour l'augmentation de données, nous avons simplement appliqué une normalisation par histogramme, puis un effet miroir horizontal et vertical. Comme nous utilisons un modèle pré-entraîné qui utilise 3 canaux, nous avons dupliqué 2 canaux pour leur appliquer une augmentation de luminosité et un filtre pour extraire plus de contours. Malgré cela, le modèle était toujours sujet à un surapprentissage. C'est la raison pour laquelle nous avons gardé uniquement les 45 premières couches du modèle pré-entraîné Xception, tout en redéfinissant les dernières couches du modèle Xception pour faciliter la réalisation du Grad-CAM. Nous avons finalement ajouté une couche Dense de 4 unités pour faciliter la détection des 4 types de classes différentes, suivie d'une couche Dense de 2 unités avec une activation softmax pour la prédiction probabiliste.

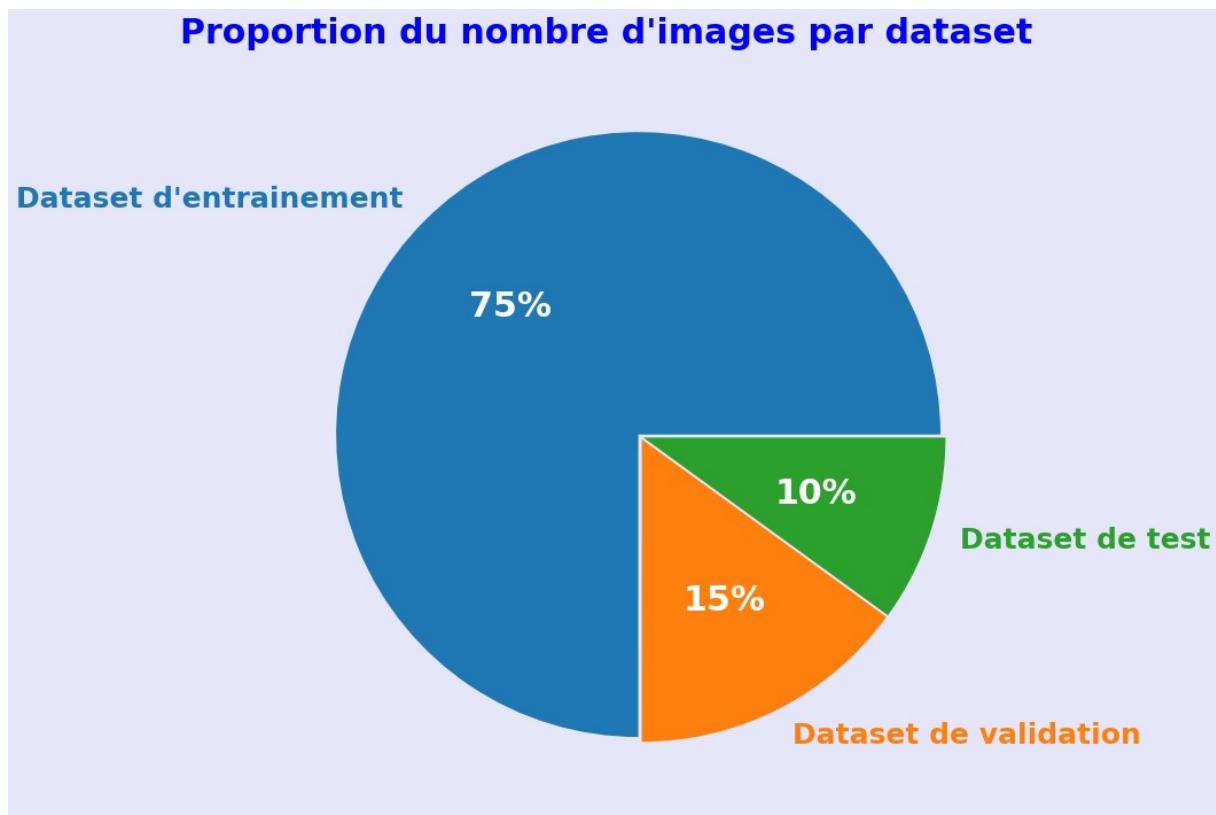
Le modèle Xception (Extreme Inception) est un réseau de neurones convolutifs profonds à architecture modulaire, inspiré de la famille des réseaux Inception, qui utilise des convolutions séparables en profondeur pour améliorer l'efficacité de calcul. Les 45 premières couches du modèle Xception sont des couches de convolution séparable en profondeur, qui sont des blocs de convolution qui séparent le traitement spatial et le traitement en profondeur des données d'entrée. Cette technique de séparation des convolutions spatiales et des convolutions en profondeur permet de réduire considérablement le nombre de paramètres du modèle tout en conservant sa capacité à apprendre des caractéristiques complexes. Le modèle Xception utilise également des blocs résiduels, comme dans le modèle ResNet, pour faciliter la formation de réseaux plus profonds et plus précis. Les blocs résiduels permettent de résoudre le problème de la vanishing gradient qui se produit lorsque le gradient devient de plus en plus petit à mesure que l'on remonte dans les couches d'un réseau profond. En résumé, les 45 premières couches du modèle Xception sont des couches de convolution séparable en profondeur avec des blocs résiduels, ce qui en fait un modèle très efficace en termes de calcul et capable d'apprendre des caractéristiques complexes pour une précision accrue.

Notre modèle est donc une combinaison de différents modèles, avec un prétraitement d'image pour le redimensionnement, la normalisation et la multiplication avec la sortie du modèle U-Net. Le modèle U-Net est chargé depuis un fichier pré-entraîné, et son architecture est utilisée jusqu'à la couche de convolution 'conv2d\_152'. Ensuite, la sortie de cette couche est utilisée comme entrée pour le modèle Xception qui a également été pré-entraîné sur ImageNet. Le modèle Xception est tronqué à la couche 45 pour obtenir une sortie intermédiaire. Ensuite, une couche convolutive, suivie d'une normalisation, d'un dropout et d'une activation relu sont appliquées avant d'être envoyées dans une autre couche convolutive, suivie de normalisation, d'un dropout et d'une activation relu. Ensuite, une couche de pooling globale, suivie d'une couche dense sont appliquées pour réduire la dimensionnalité des caractéristiques. Enfin, une couche dense de sortie avec une fonction d'activation softmax est ajoutée pour prédire les classes.

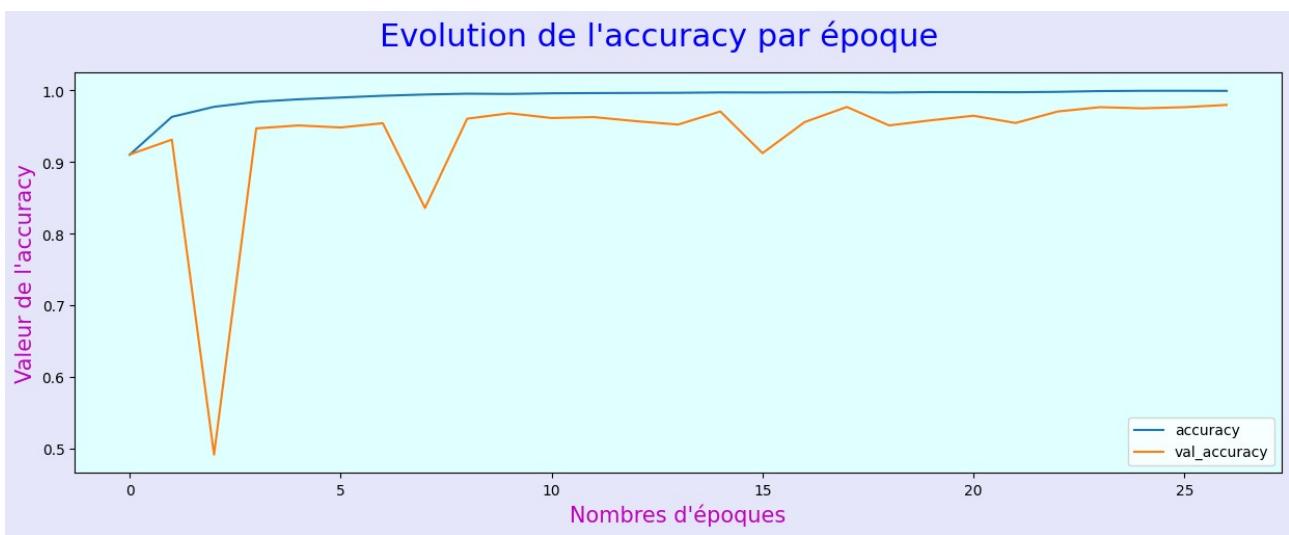
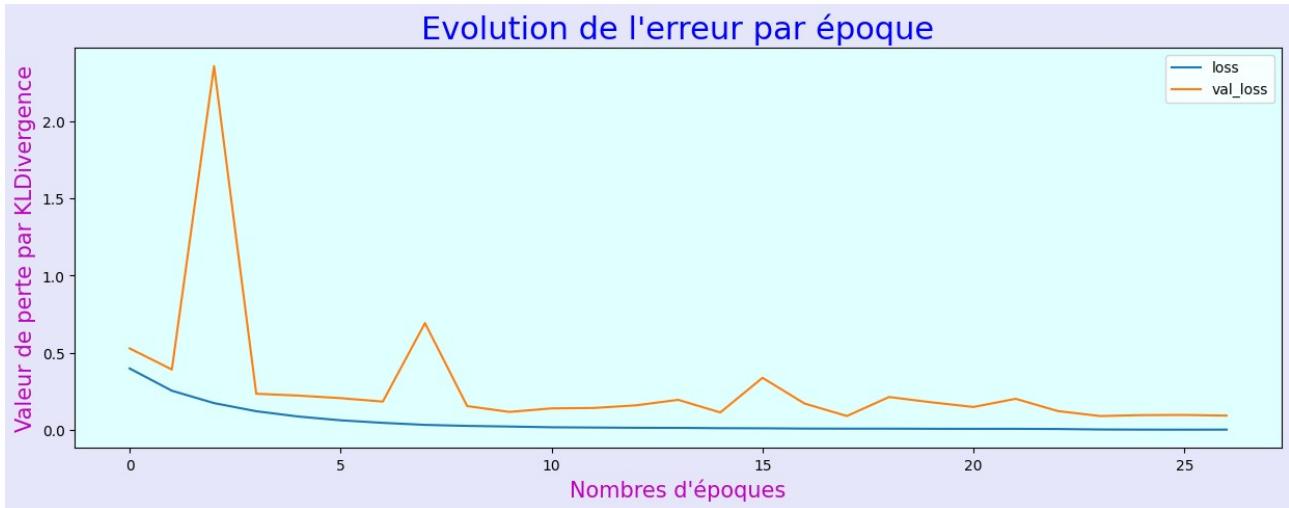
Le schéma de l'architecture peut être vu à la page suivante.



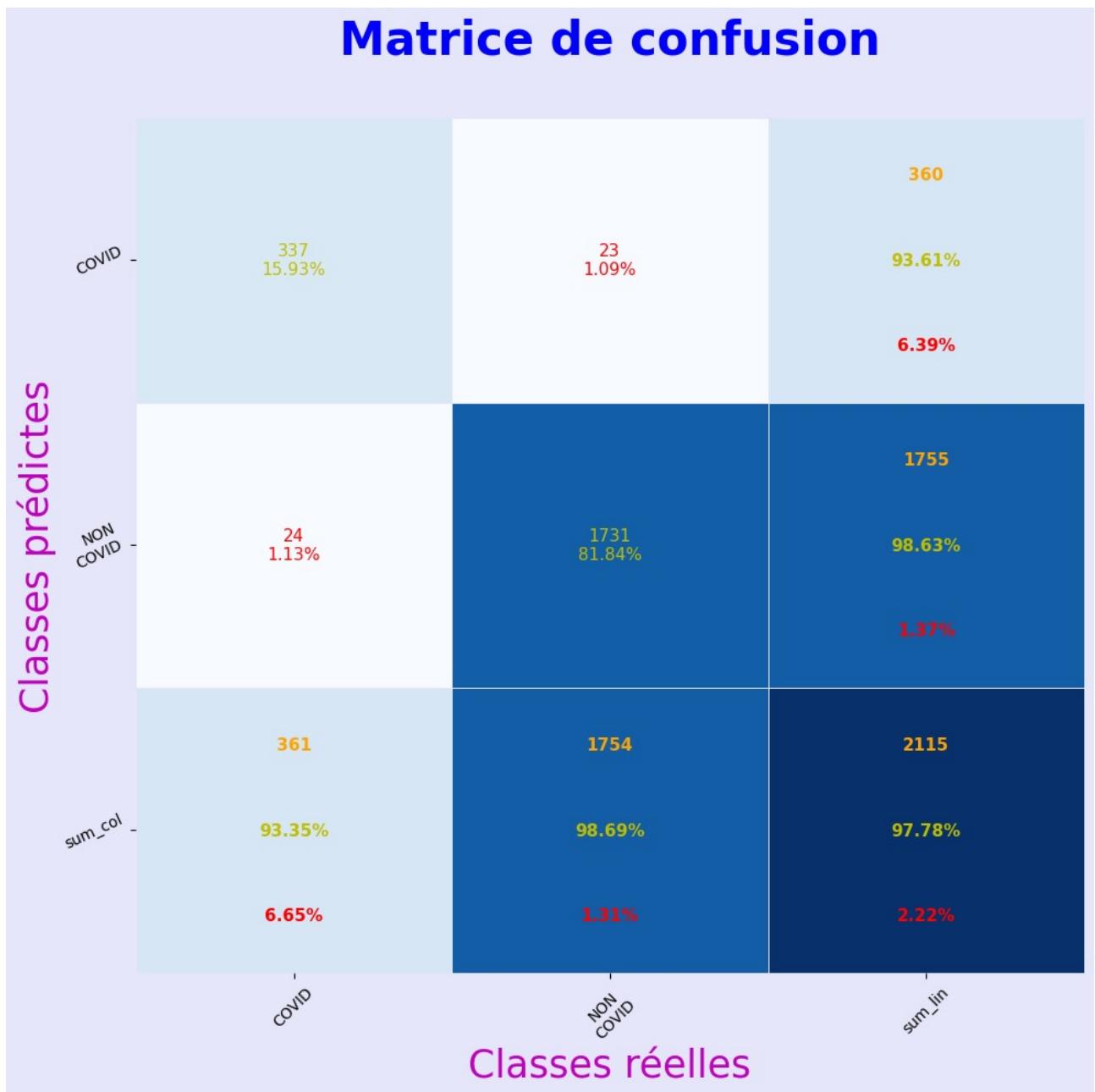
Pour limiter le surapprentissage toujours présent, nous avons séparé nos données en trois sous-dossiers : un dataset d'entraînement contenant 15873 images, un dataset de validation contenant 3177 images et un dataset de test contenant 2115 images. Voici la représentation de la proportion d'images entre chaque dataset :



Une fois l'entraînement terminé, nous obtenons un score d'environ 99 % sur les données d'entraînement et proche de 98 % sur les données de validation :



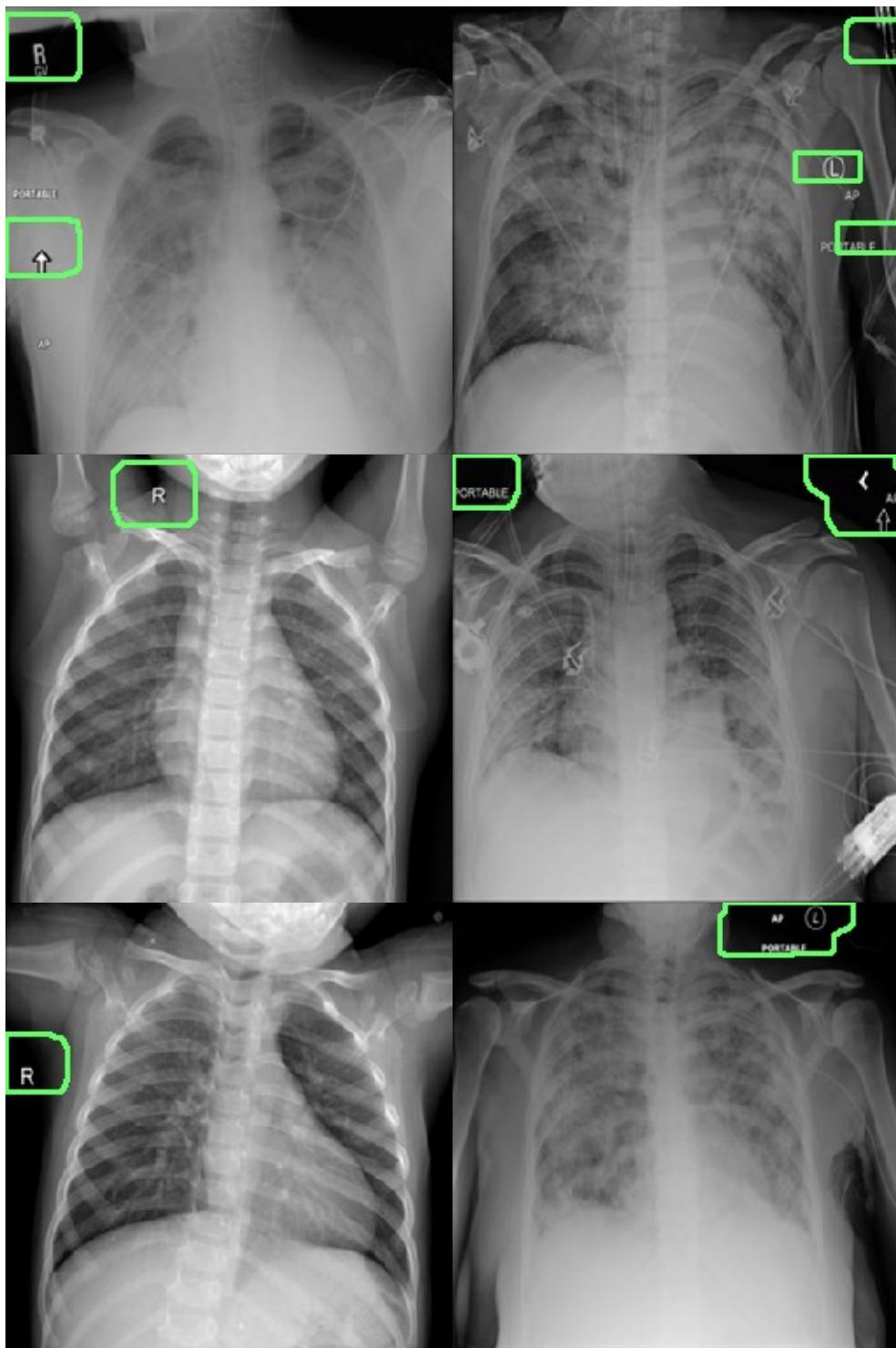
On affiche alors la matrice de confusion des prédictions effectuées sur les données de test :



Comme on peut le voir, nous obtenons une précision globale de 97,78%. On prédit 93,61 % des cas COVID, avec une erreur de 6,39 % et 98,63 % des cas non-COVID avec une erreur de 1,37 %.

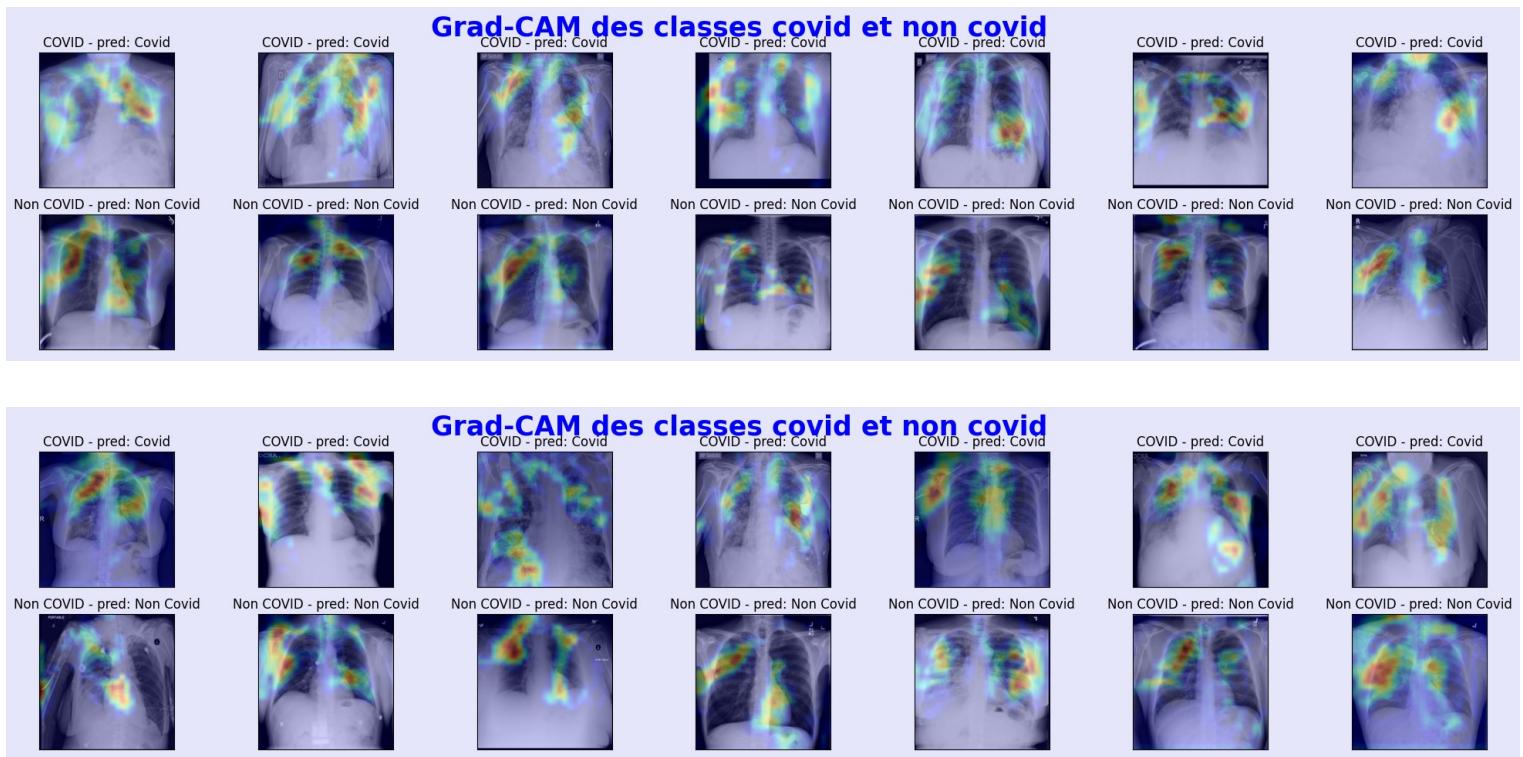
#### 4. Visualisation des zones d'intérêt grâce à Gradcam

Nous avons également développé un modèle GradCam afin de pouvoir visualiser sous forme de heatmap les zones d'intérêt du modèle, afin de localiser les ce qui semble retenir son attention pour tel ou tel label. Initialement, nous avons appliqué un gradcam sur un des premiers modèles CNN qui avait été entrainé à reconnaître les cas « COVID contre tous » (classification binaire) sans que les masques ne soient appliqués. Après utilisation de l'algo de GradCam, nous avons affiché les zones sur lesquelles le modèle semblait se concentrer pour détecter le label « COVID ». Les résultats sont affichés ci-dessous



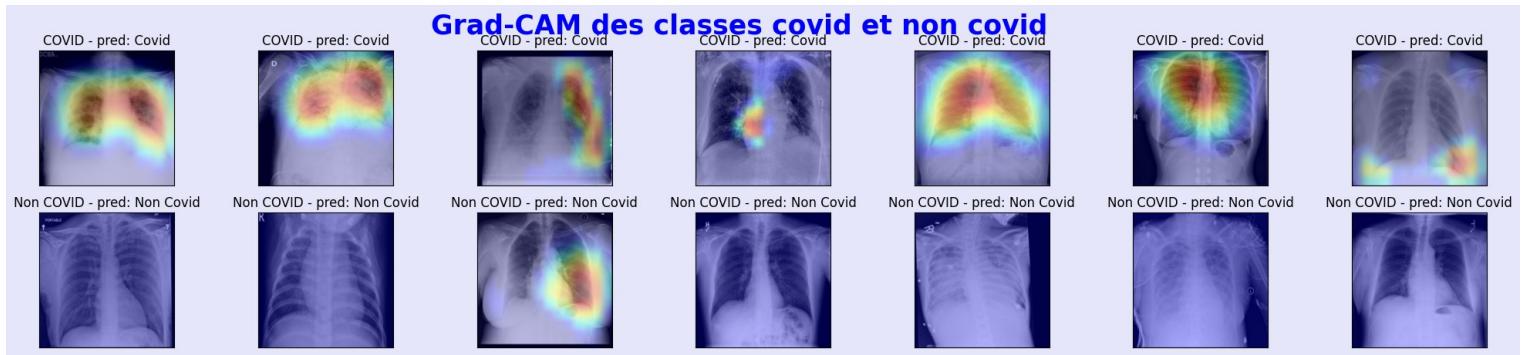
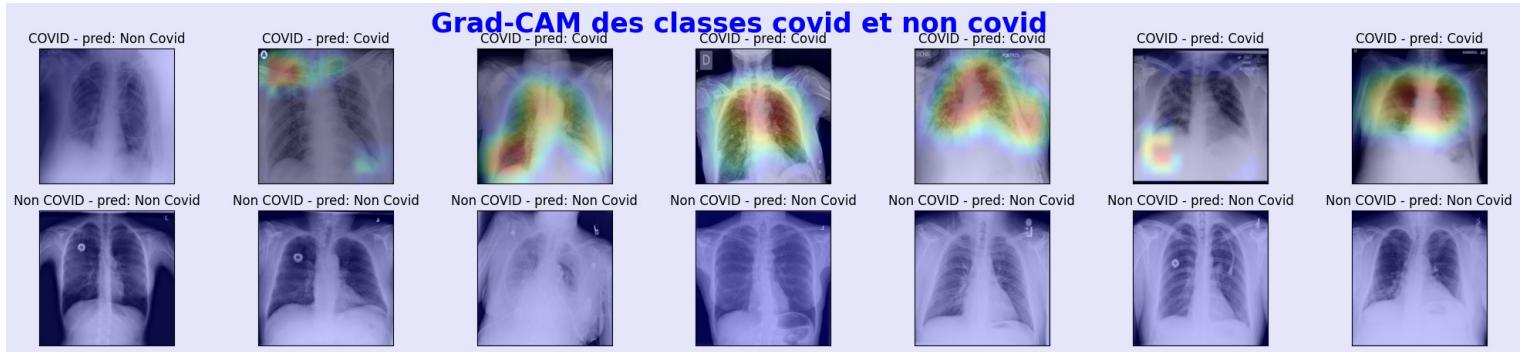
Il semble que pour le label COVID, le modèle se concentre uniquement sur des inscriptions qui apparaissent à l'image. Nous ne savons pas ce que signifient ces inscriptions, et nous n'avons pas investigué si celles-ci apparaissaient ou non sur les radios des autres labels, mais il semblerait que cette découverte justifie totalement l'emploi du modèle U-net pour la segmentation des poumons. En effet, le fait de ne garder que la partie relative aux poumons sur la radio permet de totalement supprimer le risque de voir notre modèle se concentrer sur des artefacts externes, telles que des annotations spécifiques à certaines sources.

Par la suite, nous avons fait évoluer le gradcam sous la forme d'une heatmap, coloriant ainsi les zones de l'image ayant eu, pour le modèle, la plus forte contribution dans le choix du label prédit en sortie. Nous avons fait le choix d'utiliser le gradient d'ordre 2 au lieu du gradient simple, et de clipper les valeurs de ce gradient entre 0 et 1, afin de ne pas afficher les valeurs négatives (sans intérêt pour nous dans ce cas). On peut voir ci-dessous les images générées par le gradcam pour le modèle CNN à 95 % d'accuracy présenté précédemment :



on peut dire que les radios classées Covid fournissent des résultats plus denses en terme d'information, mais la différence n'est pas forcément frappante. Il semble que le modèle soit donc bien entraîné pour détecter le Covid au sein des poumons, et qu'il se focalise notamment sur les bords des poumons, ainsi qu'au centre, partiellement. Il nous est cependant difficile de comprendre à l'œil nu ce qu'il semble avoir détecté. Aussi, il semble que le modèle se concentre sur des zones en dehors des poumons, malgré l'application du masque dans les premières couches du modèle de prédiction. Nous ne savons pas pourquoi de tels phénomènes apparaissent, mais nous pouvons clairement remettre en cause l'utilité du GradCam pour ce modèle, car les zones d'intérêt sont très dispersées et difficiles à interpréter.

Appliquons à présent le Grad-CAM sur les données de test avec le modèle 2 (Fine-tuned)



Comme on peut le voir très clairement, la heatmap est présente au niveau des poumons sur les cas COVID et généralement complètement absente des cas Non-COVID. Cela laisse penser que le modèle fait de la détection de COVID et non pas une simple classification.

On constate aussi que les résultats sont plus « harmonieux » au niveau des formes. Nous pensons que cela est dû aux différences en termes de compléxité au niveau des algorithmes : le modèle Xception est capable de repérer plus de détails et va ainsi considérer un nombre de zones d'intérêt plus élevé dans l'image, à l'inverse du CNN qui aura une tendance à s'intéresser à des zones de l'image plus ponctuelles, de par son nombre de filtres plus limité.

# Conclusion

## 1. Bilan et Interprétation des résultats

Nos modèles CNN et Xception + classifier donnent des résultats très satisfaisants. À 95 % d'accuracy pour le premier, et presque 98 % pour le second, nous sommes satisfaits des performances obtenues. Notre objectif personnel dans ce projet était double car nous souhaitions présenter deux méthodes : réaliser un modèle « naïf » from scratch de type CNN pour prouver que nous maîtrisons la réalisation d'un projet de deep learning de bout en bout, mais aussi effectuer le finetuning d'un modèle pré-entraîné tel que le Xception pour nous faire une idée des scores maximaux obtenables. Concrètement, nous souhaitions développer un modèle ayant les performances les plus élevées possibles, compte tenu de nos connaissances obtenues lors de la formation au sein de DataScientest.

Dans une logique d'une mise en production théorique, nous nous sommes également intéressés au ratio performances/stockage qui se pose lors de tout projet de ce type. Dans ce sens, nous sommes également très satisfaits des modèles proposés ici, plutôt légers (environ 7 Mo et 67 Mo pour le CNN et le Xception, respectivement). Même si nous avons conscience que ce paramètre intervient peu dans le cadre de ce sujet, nous pensons que cette notion reste globalement importante car ces architectures pourraient être employées pour d'autres tâches, dans des conditions d'utilisation totalement différentes.

Nous avons également accordé une réelle importance à l'interprétabilité des résultats : nous ne souhaitons pas obtenir en sortie un simple label, mais plutôt un score, qui témoignerait du niveau de certitude du modèle quant à la prédiction effectuée. De plus, un simple chiffre n'étant pas en soi très « parlant », nous avons joint l'exécution d'un algorithme de GradCam, permettant ainsi à l'utilisateur de visualiser les détails contenus dans chaque radio qui avaient influencé le choix du modèle. Cette démarche nous semble nécessaire, car dans le cadre de la santé, il est logique qu'un diagnostic soit effectué par un spécialiste humain plutôt que par une machine. En joignant la heatmap du GradCam au score retourné, le modèle devient alors ni plus ni moins qu'un simple outil permettant de donner un « pré-diagnostic », qui peut alors être soumis au médecin qui validera ou invalidera la prédiction du modèle, en regardant sur la radio les zones qui auront été détectées comme symptomatiques du Covid. De plus, les différences dans les résultats obtenus avec les deux modèles justifie l'idée de laisser le choix à l'utilisateur : le CNN pour les performances de vitesse de calcul et de taille, et le Xception pour les performances de certitude et de précision pour localisation des zones d'intérêt au sein de la radio.

Toujours sur le Grad-CAM, nous sommes fiers d'avoir pu déceler le biais qui était créé par la non segmentation des poumons au sein des images. En effet, le fait que le modèle puisse se concentrer sur des symboles apparaissant sur les radios nous a conforté dans l'idée que, une fois cette segmentation réalisée, il n'existe plus alors aucun biais possible dans le diagnostic effectué par le modèle. Nous nous permettons alors d'émettre un avis critique sur différents modèles et notebooks que nous avons pu voir sur le site kaggle, voir même dans des articles universitaires, qui utilisaient ce même dataset mais ne masquaient pas les images. Certains résultats, parfois inférieurs aux nôtres, sont alors à remettre en question :

ces modèles fonctionneraient-ils aussi bien sur des images segmentées ? Nous pensons que ce n'est pas le cas.

Le principal obstacle que nous avons rencontrés s'est porté sur le dataset en lui-même : à la fois riche (environ 21000 images) et en même temps trop déséquilibré pour pouvoir facilement entraîner et généraliser un modèle de deep-learning, nous avons suivi de nombreuses pistes différentes pour pouvoir utiliser au maximum l'ensemble des features de ce dataset. Initialement partis sur des méthodes d'oversampling/undersampling coûteuses en termes de gestion de la mémoire vive, nous avons rapidement saisi tout l'intérêt de nous intéresser l'équilibre des labels au sein d'un batch. Hélas, la librairie tensorflow ne nous permettant pas de facilement avoir un droit de regard sur cet aspect de la mise en forme des données, nous sommes fiers d'avoir réalisé le générateur que nous avons présenté dans la partie « gestion des labels » (partie 1 – 4). En effet, nous avons observé un gain de performance conséquent, et nous pensons que c'est aujourd'hui la meilleure méthode que nous ayons trouvée pour à la fois réaliser l'exploitation complète du dataset, tout en optimisant la significativité en termes d'information de chaque batch présenté au modèle durant l'entraînement. Cette méthode, couplée aux possibilités fournies par l'utilisation d'un TFRecordDataset et des méthodes d'augmentation des données au sein de Tensorflow, nous a permis d'atteindre l'objectif en terme de performances que nous nous étions fixés.

Au sujet des performances, les scores respectifs de nos deux modèles ont été calculés sur des datasets de test non équilibrés, à l'inverse de ceux utilisés pour l'entraînement. En effet, nous ne souhaitions pas établir un biais en testant les modèles sur des sets de test générés par notre générateur, qui aurait alors dupliqué de nombreux exemples de la catégorie covid, ce qui aurait eu pour effet de propager de multiples fois une même erreur et réduire le score final (biais pessimiste), ou à l'inverse des bonnes réponses ce qui aurait eu pour effet d'augmenter le score final (biais optimiste). Nos scores sont donc basés sur la répartition des labels au sein du dataset, soit environ 16 % de labels Covid, et 84 % de labels non-Covid. Nous aurions aimé créer un dataset de test encore plus représentatif de la réalité en adaptant ces proportions aux probabilités, pour un spécialiste, de rencontrer un cas Covid dans le monde réel, mais hélas nous n'avons pas trouvé comment obtenir cette information.

## 2. Améliorations possibles et autres pistes de réflexion

Aujourd'hui, nous pensons que les axes suivants pourraient apporter une amélioration au projet :

- Utilisation des deux modèles obtenus pour faire de l'extraction de features, et combiner les informations obtenues au sein d'un même réseau classifier. Il serait également possible d'utiliser d'autres modèles que ceux qui ont été présentés ici.
- Généralisation du problème à 4 classes et plus. Une idée retenue, au début des essais sur le CNN, était de réaliser une classification à 3 classes : Covid, Normal et Pneumonia/Lung Opacity. Cette classification a pour intérêt de distinguer la classe normale (non malade) des cas malades (pneumonie, etc.) sans toutefois les confondre avec le Covid.

- Utilisation d'une custom-loss, et plus précisément d'une focal-loss adaptée.
- Augmenter la taille du dataset en récupérant d'autres banques d'images de radiographie, et les labeliser correctement. Notre U-net étant extrêmement précis, nous n'aurions alors plus besoin des masques qui ont été initialement fournis avec le dataset de départ : nous pouvons à présent les calculer sans les connaître au préalable.

Parmi les pistes intéressantes que nous n'avons pas exploré, nous pouvons citer l'optimisation PSO (particule swarm optimization) qui semble particulièrement adaptée pour l'extraction de features à partir de modèles pré-entraînés.

### 3. Remerciements

Nous souhaitons remercier :

- Notre mentor de projet, Okba Bentoumi, pour son soutien et ses nombreux conseils.
- DataScientest, pour les cours de qualité et les bonnes méthodes
- Maëlys de DataScientest, pour son temps et son implication
- Lucas Rousse pour son aide à la rédaction de ce rapport
- Cécile Chiron qui a été d'une patience extrême envers Mr Pagesy
- Naruto et ses 220 épisodes, parfaits pour rester éveillé pendant les entraînements de modèles