International Journal of Innovation and Applied Studies
ISSN 2028-9324 Vol. 38 No. 4 Feb. 2023, pp. 860-872

http://www.ijias.issr-journals.org/

Système de détection de la pneumonie dans les images radiographiques thoraciques

[System for pneumonia detection in chest X-ray images]

ILUNGA MBUYAMBA Elisée and MUSENGA SHAMBUYI Plamedi

Institut Supérieur de Techniques Appliquées (ISTA), Kinshasa, RD Congo

ABSTRACT: The dark and worrying health picture characterized by the revelations made by international organizations concerning pneumonia challenged us as researchers. It is in this perspective that we decided to make our contribution to this problem by presenting a system for pneumonia detection in chest x-ray images. To achieve this goal, we used an approach based on deep learning to properly identify pathological or non-pathological radiographs by setting up a convolutional neural network called Xception. Two optimization algorithms were selected, namely Adam and SGD. The setting of hyperparameters of our convolutional neural network led us to a promising result compared to the size of our dataset. In conclusion, the obtained results in our experiments showed that the SGD optimization algorithm reached the best result of 92% accuracy on new data with a learning rate of 0.001 for 20 epochs.

KEYWORDS: Pneumonia, CNN, Xception, SGD, x-ray.

RESUME: Le tableau sanitaire sombre et inquiétant caractérisé par les révélations faites par les organisations internationales concernant la pneumonie nous a interpellés en tant que chercheur. C'est dans cette perspective que nous avons décidé d'apporter notre contribution à cette problématique en présentant un système de détection des pneumonies sur les images radiographiques thoraciques. Pour atteindre cet objectif, nous avons utilisé une approche basée sur le deep learning pour bien identifier les radiographies pathologiques ou non pathologiques en mettant en place un réseau de neurones convolutifs appelé Xception. Deux algorithmes d'optimisation ont été sélectionnés, à savoir Adam et SGD. Le paramétrage des hyperparamètres de notre réseau de neurones convolutifs nous a conduit à des résultats prometteurs par rapport à la taille de notre jeu de données. En conclusion, les résultats obtenus dans nos expériences ont montré que l'algorithme d'optimisation SGD a atteint le meilleur résultat de 92% de précision sur les nouvelles données avec un taux d'apprentissage de 0,001 pour 20 époques.

MOTS-CLEFS: Pneumonie, CNN, Xception, SGD, Radiographie.

1 INTRODUCTION

La pneumonie est la première cause infectieuse de mortalité chez l'enfant. En 2017, 808 694 d'enfants de moins de 5 ans sont décédés d'une pneumonie, soit 15% des décès dans ce groupe d'âge à l'échelle mondiale. Elle affecte les enfants et les familles partout dans le monde, mais sa prévalence est plus forte en Asie du Sud et en Afrique subsaharienne [(1)].

1 https://www.who.int/fr/news-room/fact-sheets/detail/pneumonia

Corresponding Author: ILUNGA MBUYAMBA Elisée

Henrietta Fore, directrice générale de l'UNICEF, a déclaré : « Chaque jour, près de 2 200 enfants de moins de 5 ans meurent de pneumonie, une maladie que l'on sait pourtant soigner et éviter dans la majorité de cas ». En Afrique, entant qu'une première cause infectieuse, la pneumonie devient de plus en plus un danger permanent sur la santé de nombreux enfants [(2)].

La République Démocratique du Congo est le quatrième pays au monde parmi les pays les plus touchés par la pneumonie. Après le Nigeria, l'Inde et le Pakistan. En 2016, cette maladie a causé la mort de 29 500 enfants de moins de 5 ans, en 2018, cette maladie a causé la mort de près de 40 000 enfants de moins de 5 ans. Et chaque année près de cinq enfants âgés de moins de 5 ans meurent chaque heure de la pneumonie en RDC.

Selon le plan Global de lutte contre la pneumonie (GAPPD), il est possible de réduire le taux de mortalité enfantine lié à cette maladie en République Démocratique du Congo. Et cette réduction du taux de mortalité enfantine passe par un bon diagnostic afin d'une prise en charge adéquate.

L'administration de soins convenables dépend du niveau de formation du personnel de santé à diagnostiquer et à détecter correctement la pneumonie. Ce qui permettra une bonne prise en charge et un traitement adapté. Or, en République de Démocratique du Congo, outre dans des grandes villes, nous avons de sérieux problèmes pour identifier et traiter la pneumonie à cause du manque de formation du personnel soignant et de l'insuffisance des équipements médicaux appropriés.

Selon une étude menée par le PATH sur la pneumonie en RDC, approximativement 40 pour cent des prestataires de santé interrogés n'avaient jamais reçu de formation sur la pneumonie et que les erreurs d'interprétations des clichés radiographiques étaient l'un des obstacles pour un bon diagnostic de la pneumonie [(3)]. C'est dans cette optique que nous avons jugé bon de mettre en place un système de détection de la pneumonie via les images radiographies thoraciques.

2 MATÉRIELS ET MÉTHODES

Il est difficile à tout chercheur de réaliser un travail scientifique sans pour autant faire recours à certains procédés intellectuels pouvant bien orienter son objet d'étude, il s'agit donc des matériels et méthodes de recherches.

2.1 MATÉRIELS

Dans cette partie, nous présenterons le dataset que nous utiliserons et les outils qui nous permettront de mettre en place le système de détection de la pneumonie [(4)]. Après avoir présenté le dataset et les outils que nous utiliserons, nous introduirons une variante du CNN[(5)] appelé « Xception ».

2.1.1 PRÉSENTATION DU JEU DE DONNÉES

Dans le cadre de notre travail, nous utilisons un dataset de la plate-forme web Kaggle[(6)] pour entraîner notre modèle. L'ensemble de données est organisé en 2 dossiers, le TRAIN_SET et le TEST_SET. Ces dossiers contiennent à leur tour deux sous-dossiers, un sous-dossier contenant les images pathologiques et un autre sous-dossier contenant les images normales (voir la figure 1).

Nous avons fait recours à cette plate-forme web à cause du nombre des clichés radiographiques que nous avions besoin pour entraîner notre modèle. Cet ensemble des données contient 5 863 images radiographiques (JPEG) et 2 catégories (Pathologique /Normale).

2 SIMON Jean-Jacques, https://www.unicef.org/drcongo/communiques-de-presse/pneumonie-plusieurs-organismes-tirent-sonnette-alarme

3 Analyse du paysage du Marché de la pneumonie en République Démocratique du Congo (RDC), USA, 2018, p.21

4 https://www.who.int/fr/news-room/fact-sheets/detail/pneumonia.

5 Nikhil Buduma and Nicholas Lacascio, Fundamentals of Deep Learning Designing Next-Generation Machine Intelligence Algorithms, Usa, 2017, p.81

6 https://www.kaggle.org

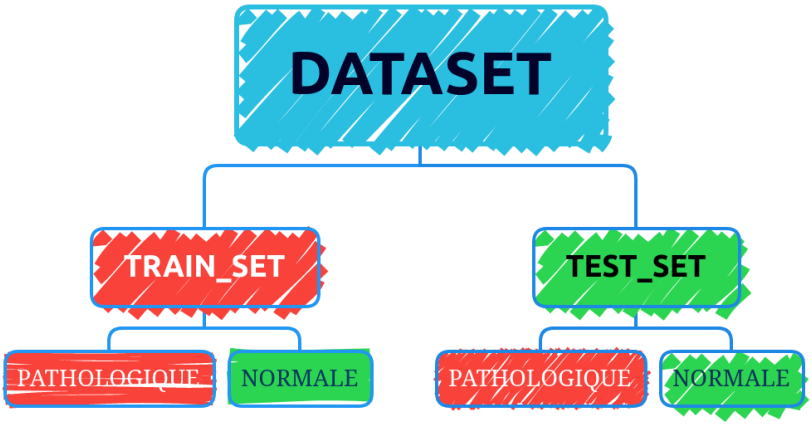

Fig. 1. Représentation du jeu de données

2.1.2 XCEPTION[7]

Xception est une architecture de réseau de neurones à convolution profonde qui implique des convolutions séparables en profondeur. Il a été développé par les chercheurs de Google. Google a présenté une interprétation des modules Inception dans les réseaux de neurones convolutifs comme étant une étape intermédiaire entre la convolution régulière et l'opération de convolution séparable en profondeur (une convolution en profondeur suivie d'une convolution ponctuelle). Dans cette optique, une convolution séparable en profondeur peut être comprise comme un module Inception avec un nombre maximal de tours. Cette observation les amène à proposer une nouvelle architecture de réseau de neurones à convolution profonde inspirée d'Inception, où les modules Inception ont été remplacés par des convolutions séparables en profondeur.

2.1.2.1 L'ARCHITECTURE XCEPTION

Une autre variante de l'architecture GoogLeNet[8] est également à noter : Xception (qui signifie Extrême Inception) a été proposé en 2016 par François Chollet (l'auteur de Keras), et il a largement surpassé Inception-v3 sur une énorme tâche de vision (350 millions d'images et 17 000 classes). Tout comme Inception-v4, il fusionne également les idées de GoogLeNet et ResNet, mais il remplace les modules de création par un type de couche appelé convolution séparable en profondeur (ou convolution séparable tout court). Ces couches avaient déjà été utilisées dans certaines architectures CNN, mais elles n'étaient pas aussi centrales que dans l'architecture Xception. Alors qu'une couche convolutive régulière utilise des filtres qui tentent de capturer simultanément des motifs spatiaux (par exemple, un ovale) et modèles de canaux (par exemple, bouche + nez + yeux = visage), une couche convolutive séparable fait l'hypothèse forte que les modèles spatiaux et les modèles transcanaux peuvent être modélisés séparément (tel qu'illustré dans la figure 14). Ainsi, il est composé de deux parties : la première partie applique un seul filtre spatial pour chaque carte d'entités en entrée, puis la deuxième partie regarde exclusivement pour les modèles cross-canal - c'est juste une couche convolutive régulière avec filtres 1 × 1.

2.2 MÉTHODES

D'après MADELEINE, GRAWITZ, la méthode est l'ensemble des opérations intellectuelles par lesquelles une discipline cherche à atteindre les vérités qu'elle poursuit, les démontre, les vérifie[9].

Pour élaborer ce travail, nous avons commencé par :

Importer les modules python pour l'apprentissage automatique et la datascience que nous utiliserons dans notre système.

7 CHOLLET, François, Xception: Deep Learning with Depthwise Separable Convolutions, 2016
8 Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, canada, O'Reilly Media, p.368, 2017.
9 M.GRAWITZ, Méthode en sciences sociales, Paris, Dalloz 1986, p.15

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

Une fois l'importation terminé, nous créons une fonction pour filtrer les images corrompues. Celle-ci ne prend aucun paramètre, elle a une variable qui est initialisé à 0, cette variable nous aide à comptabiliser le nombre des images corrompues.

Après avoir filtré le dataset, nous générons un nouveau dataset à partir de ce dernier. Celui-ci est également subdiviser en deux parties, le train_set et le validation_set.

Nous créons une fonction python pour visualiser les images contenues dans notre dataset. Pour ce faire, nous utilisons Matplotlib, une bibliothèque python pour le traçage et visualisaion des données sous formes de graphiques.

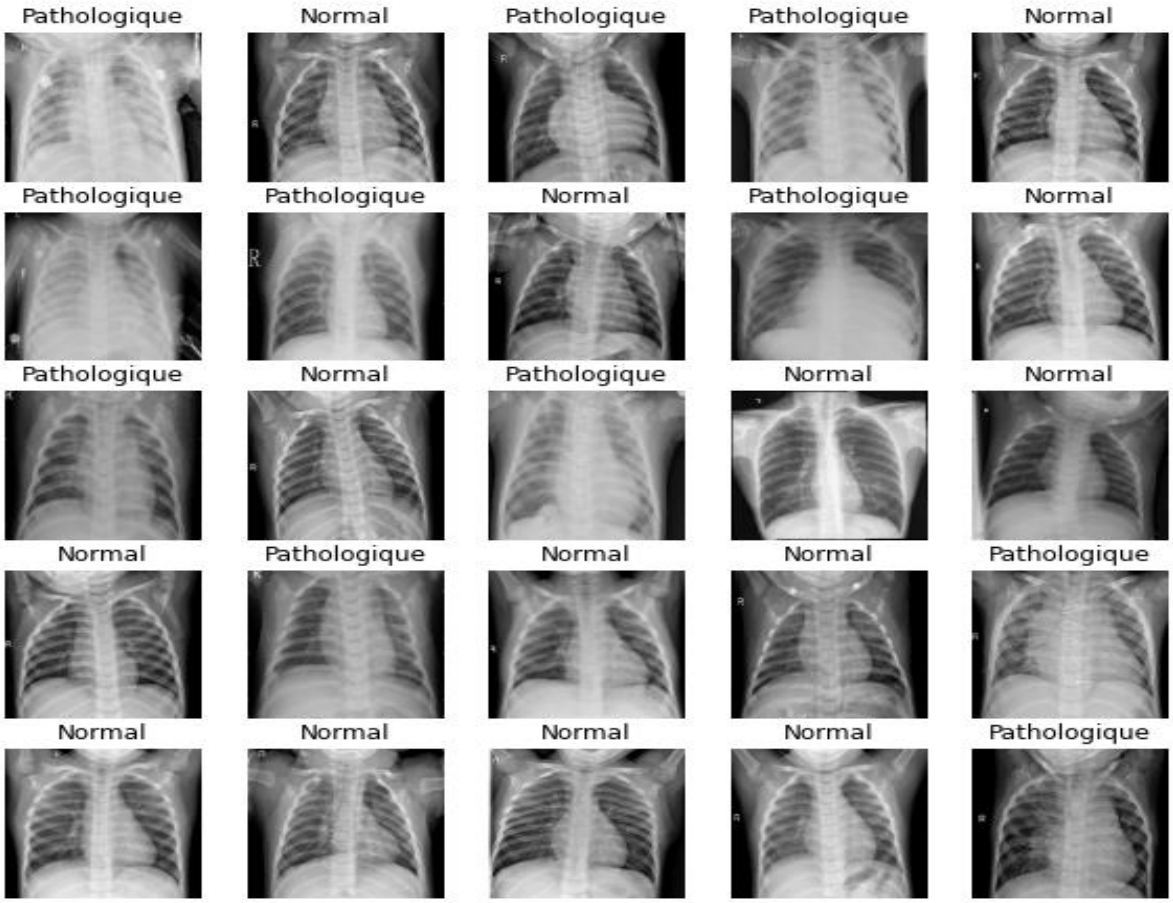

Fig. 2. Visualisation de notre dataset

Après cette étape de visualisation, nous allons maintenant faire le pré-traitement de nos données.

Comme nous ne disposons pas d'un grand ensemble de données nous appliquons des transformations aléatoire [10] à notre jeu de données d'entraînement, ces transformations permettent d'exposer notre modèle à différents aspects des données d'entraînement tout en ralentissant le surajustement.

10 Ian Goodfellow et all, Deep Learning, 2016, p.240

```
data_augmentation = keras.Sequential(
      [
            layers.experimental.preprocessing.RandomFlip("horizontal"),
            layers.experimental.preprocessing.RandomRotation(0.1),
      ]
)
```

Le pré-traitement nous aidera à redimensionner les valeurs d'entrée dans une nouvelle plage de valeurs. Pour ce faire nous allons définir une variable nommée inputs, dans laquelle nous appelons la classe keras.Input qui prendra comme paramètre une matrice à 3 démenions représentant la taille d'une image d'entrée.

```
inputs = keras.Input(shape=(180, 180, 3))
```

Nous définissons une autre variable x pour stocker la classe layers.experimental.preprocessing.Rescaling dans laquelle nous passons 1./255 comme paramètre enfin de redimensionner nos images d'entrée en une valeur compris dans l'intervalle fermée de [0, 1].

```
x = data_augmentation(inputs)
x = layers.experimental.preprocessing.Rescaling(1./255)(x)
```

Enfin, nous mettons en place un modèle prédéfinie d'apprentissage profonde Xception pour notre travail. Notre modèle sera contenu dans une fonction python nommée make_model. Celle-ci prend deux paramètres, le premier est input_shape et le second num_classes.

3 RÉSULTATS

Nous ajusterons deux hyperparamètres [11] de notre modèle pour trouver un excellent modèle. Ce deux hyperparamètres sont l'algorithme d'optimisation [12] et le taux d'apprentissage ou le learning rate [13].

Nous commencerons l'entraînement par l'algorithme d'optimisation Adam et un learning rate de 0.001, 0.01 et 0.5 pour 30 epoch, nous visualiserons les résultats pour des données d'entraînement et de validation. Nous terminerons avec un autre algorithme d'optimisation qui est le SGD et un learning rate de 0.001 pour 20 epoch, nous visualiserons une fois de plus les résultats pour des données d'entraînement et de validation.

3.1 L'ALGORITHME D'OPTIMISATION ADAM [14]

L'algorithme d'optimisation d'Adam est une méthode de descente de gradient stochastique basée sur l'estimation adaptative des moments du premier ordre. Selon Kingma et al., 2014, la méthode est « efficace du point de vue informatique, nécessite peu de mémoire, est invariante au redimensionnement diagonal des gradients et est bien adaptée aux problèmes volumineux en termes de données/paramètres ».

11 Opcit, p.120
12 Dimitri P. Bertsekas, Convex Optimization Algorithms, Usa, Athena Scientific, 2015, p.53.
13 Christopher M. Bishop, Pattern Recognition and Machine Learning, Californie, springer, 2006, p.240
14 Opcit, p.81

3.1.1 FORMATION DU MODÈLE À L'AIDE D'ADAM AVEC UN LEARNING RATE DE 0.001

3.1.1.1 FORMATION DU MODÈLE

```
Entrée [11]: # Nous formons notre modèle avec 30 itérations
             # une learning rate de 0.001.
             epochs = 30

             callbacks = [
                 keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
             ]
             model.compile(
                 optimizer=keras.optimizers.Adam(learning_rate=0.001),
                 loss="binary_crossentropy",
                 metrics=["accuracy"],
             )
```

Fig. 3. Formation du modèle en utilisant Adam comme algorithme d'optimisation et un learning rate de 0.001

3.1.1.2 ENTRAÎNEMENT DU MODÈLE

```
Entrée [12]: my_model = model.fit(
                 train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
             )

Epoch 1/30
27/27 [==============================] - 328s 10s/step - loss: 0.4729 - accuracy: 0.7777 - val_loss: 0.7552 - val_
accuracy: 0.5116
Epoch 2/30
27/27 [==============================] - 228s 8s/step - loss: 0.3011 - accuracy: 0.8913 - val_loss: 1.0145 - val_a
ccuracy: 0.5116
Epoch 3/30
27/27 [==============================] - 214s 8s/step - loss: 0.2567 - accuracy: 0.9071 - val_loss: 1.2882 - val_a
ccuracy: 0.5116
Epoch 4/30
27/27 [==============================] - 216s 8s/step - loss: 0.2664 - accuracy: 0.8993 - val_loss: 1.8764 - val_a
ccuracy: 0.5116
Epoch 5/30
27/27 [==============================] - 227s 8s/step - loss: 0.1866 - accuracy: 0.9339 - val_loss: 2.4609 - val_a
ccuracy: 0.5116
Epoch 6/30
27/27 [==============================] - 213s 8s/step - loss: 0.2234 - accuracy: 0.8952 - val_loss: 3.3561 - val_a
ccuracy: 0.5116
Epoch 7/30
27/27 [==============================] - 220s 8s/step - loss: 0.2207 - accuracy: 0.9005 - val_loss: 3.8762 - val_a
ccuracy: 0.5116
Epoch 8/30
27/27 [==============================] - 213s 7s/step - loss: 0.2160 - accuracy: 0.9081 - val_loss: 4.7944 - val_a
ccuracy: 0.5116
Epoch 9/30
27/27 [==============================] - 213s 8s/step - loss: 0.2258 - accuracy: 0.9143 - val_loss: 4.7980 - val_a
ccuracy: 0.5116
Epoch 10/30
27/27 [==============================] - 217s 8s/step - loss: 0.1960 - accuracy: 0.9282 - val_loss: 5.5155 - val_a
ccuracy: 0.5116
Epoch 11/30
```

Fig. 4. Entraînement du modèle en utilisant Adam comme algorithme d'optimisation et un learning rate de 0.001

3.1.1.3 VISUALISATION DES RÉSULTATS

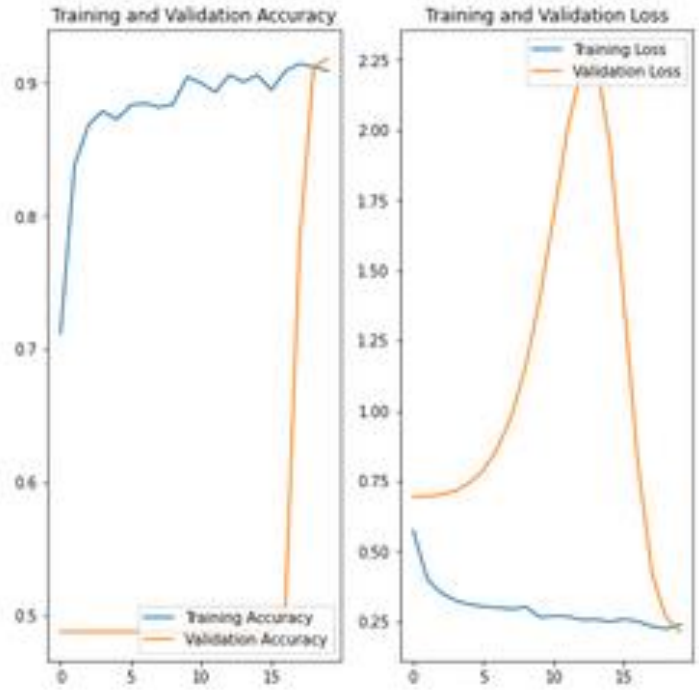

Fig. 5. Visualisation des résultats en utilisant Adam comme algorithme d'optimisation et un learning rate de 0.001

3.1.2 FORMATION DU MODÈLE À L'AIDE D'ADAM AVEC UN LEARNING RATE DE 0.01

3.1.2.1 FORMATION DU MODÈLE

Formation du modèle

Entrée [11]:
```
# Nous formons notre modèle avec 30 itérations
# une learning rate de 0.01.
epochs = 30

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
]
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.01),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
```

Fig. 6. Formation du modèle en utilisant Adam comme algorithme d'optimisation et un learning rate de 0.01

3.1.2.2 ENTRAÎNEMENT DU MODÈLE

```
Entrée [12]: my_model = model.fit(
                 train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
             )

Epoch 1/30
27/27 [==============================] - 336s 11s/step - loss: 1.0023 - accuracy: 0.7216 - val_loss: 7.6757 - val_
accuracy: 0.5116 -
Epoch 2/30
27/27 [==============================] - 225s 8s/step - loss: 0.5756 - accuracy: 0.7485 - val_loss: 1.7782 - val_a
ccuracy: 0.5116
Epoch 3/30
27/27 [==============================] - 222s 8s/step - loss: 0.4663 - accuracy: 0.8128 - val_loss: 11.6806 - val_
accuracy: 0.5116
Epoch 4/30
27/27 [==============================] - 217s 8s/step - loss: 0.4434 - accuracy: 0.8275 - val_loss: 3.2045 - val_a
ccuracy: 0.5116
Epoch 5/30
27/27 [==============================] - 211s 8s/step - loss: 0.3036 - accuracy: 0.8893 - val_loss: 0.4953 - val_a
ccuracy: 0.8233
Epoch 6/30
27/27 [==============================] - 213s 7s/step - loss: 0.3229 - accuracy: 0.8708 - val_loss: 4.0316 - val_a
ccuracy: 0.5116
Epoch 7/30
27/27 [==============================] - 214s 8s/step - loss: 0.3296 - accuracy: 0.8640 - val_loss: 1.0328 - val_a
ccuracy: 0.6093
Epoch 8/30
27/27 [==============================] - 211s 7s/step - loss: 0.3228 - accuracy: 0.8943 - val_loss: 0.6950 - val_a
ccuracy: 0.7628
Epoch 9/30
27/27 [==============================] - 212s 8s/step - loss: 0.3197 - accuracy: 0.8705 - val_loss: 2.1595 - val_a
ccuracy: 0.5674
Epoch 10/30
27/27 [==============================] - 220s 8s/step - loss: 0.3269 - accuracy: 0.8594 - val_loss: 1.6111 - val_a
ccuracy: 0.5395
```

Fig. 7. Entraînement du modèle en utilisant Adam comme algorithme d'optimisation et un learning rate de 0.01

3.1.2.3 VISUALISATION DE RÉSULTATS

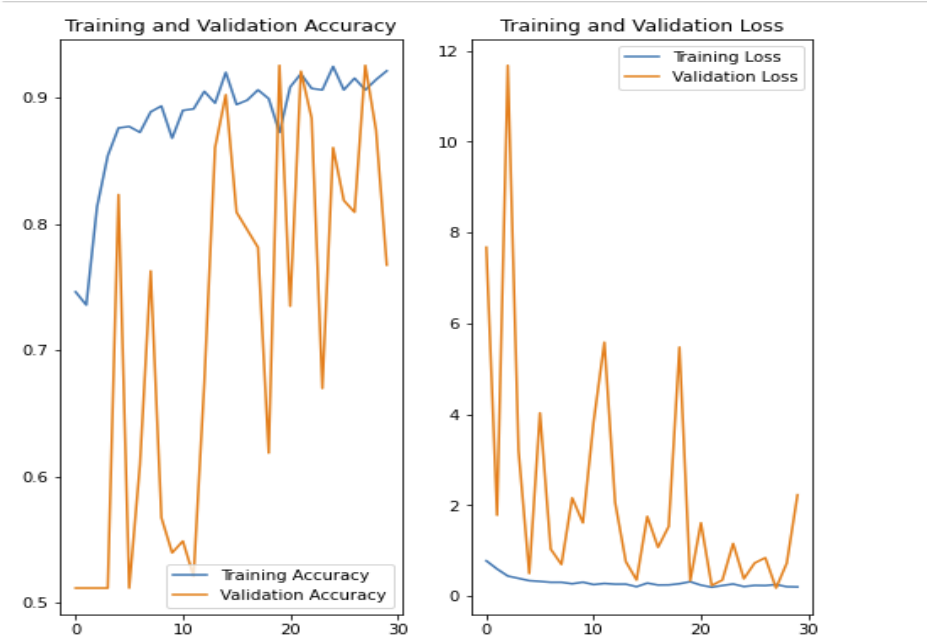

Fig. 8. Visualisation des résultats en utilisant Adam comme algorithme d'optimisation et un learning rate de 0.01

3.1.3 FORMATION DU MODÈLE À L'AIDE D'ADAM AVEC UN LEARNING RATE DE 0.5

3.1.3.1 FORMATION DU MODÈLE

```
Entrée [12]: # Nous formons notre modèle avec 30 itérations
             # une learning rate de 0.5.
             epochs = 30

             callbacks = [
                 keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
             ]
             model.compile(
                 optimizer=keras.optimizers.Adam(learning_rate=0.5),
                 loss="binary_crossentropy",
                 metrics=["accuracy"],
             )
```

Fig. 9. Formation du modèle en utilisant Adam comme algorithme d'optimisation et un learning rate de 0.5

3.1.3.2 ENTRAÎNEMENT DU MODÈLE

```
Entrée [13]: my_model = model.fit(
                 train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
             )
```

```
Epoch 1/30
42/42 [==============================] - 456s 9s/step - loss: 21.6253 - accuracy: 0.5235 - val_loss: 3825.9451 - v
al_accuracy: 0.4879
Epoch 2/30
42/42 [==============================] - 348s 8s/step - loss: 0.7501 - accuracy: 0.5753 - val_loss: 158.3917 - val
_accuracy: 0.4879
Epoch 3/30
42/42 [==============================] - 326s 8s/step - loss: 0.5995 - accuracy: 0.6780 - val_loss: 1.1470 - val_a
ccuracy: 0.5152
Epoch 4/30
42/42 [==============================] - 323s 7s/step - loss: 0.5389 - accuracy: 0.7372 - val_loss: 0.7813 - val_a
ccuracy: 0.6758
Epoch 5/30
42/42 [==============================] - 332s 8s/step - loss: 0.5772 - accuracy: 0.7199 - val_loss: 43.0439 - val_
accuracy: 0.4879
Epoch 6/30
42/42 [==============================] - 346s 8s/step - loss: 0.5282 - accuracy: 0.7376 - val_loss: 510.4443 - val
_accuracy: 0.4879
Epoch 7/30
42/42 [==============================] - 327s 8s/step - loss: 0.5327 - accuracy: 0.7514 - val_loss: 193.0438 - val
_accuracy: 0.4879
Epoch 8/30
42/42 [==============================] - 321s 7s/step - loss: 0.4948 - accuracy: 0.7600 - val_loss: 84.7816 - val_
accuracy: 0.4879
Epoch 9/30
42/42 [==============================] - 349s 8s/step - loss: 0.5301 - accuracy: 0.7662 - val_loss: 4.1127 - val_a
ccuracy: 0.5455
Epoch 10/30
42/42 [==============================] - 346s 8s/step - loss: 0.5710 - accuracy: 0.7354 - val_loss: 54.9729 - val_
accuracy: 0.5121
Epoch 11/30
```

Fig. 10. Entraînement du modèle en utilisant Adam comme algorithme d'optimisation et un learning rate de 0.5

3.1.3.3 VISUALISATION DE RÉSULTATS

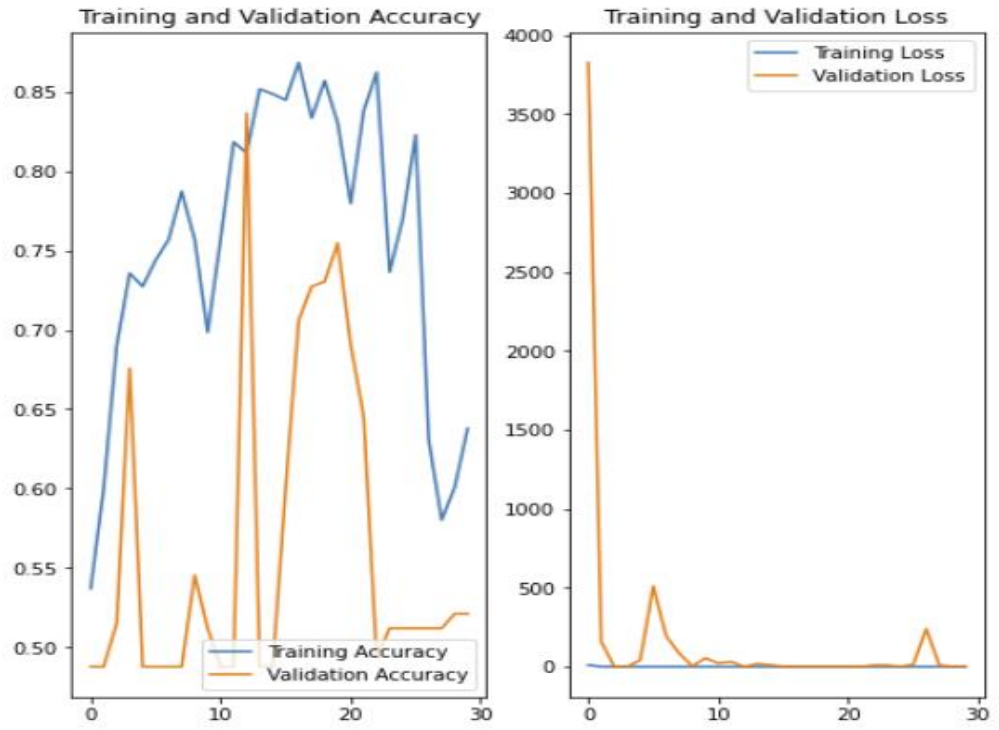

Fig. 11. Visualisation des résultats en utilisant Adam comme algorithme d'optimisation et un learning rate de 0.5

3.2 L'ALGORITHME D'OPTIMISATION SGD[15]

L'algorithme d'optimisation SGD est la stratégie d'optimisation la plus populaire et presque idéale pour les tâches d'apprentissage en profondeur. Il essaie d'apprendre le paramètre optimal (ici les poids) pour lequel le gradient de la fonction de perte est minimum. Un dégradé n'est rien d'autre que la pente, la tangente ou la valeur différenciée en ce point.

3.2.1 FORMATION DU MODÈLE AVEC UN LEARNING RATE DE 0.001

3.2.1.1 FORMATION DU MODÈLE

Entrée [11]:
```
# Nous formons notre modèle avec 20 itérations
# une learning rate de 0.001.
epochs = 20

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
]
model.compile(
    optimizer=keras.optimizers.SGD(learning_rate=0.001),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
```

Fig. 12. Formation du modèle en utilisant SGD comme algorithme d'optimisation avec un learning rate de 0.001

15 Opcit, p.120

3.2.1.2 ENTRAÎNEMENT DU MODÈLE

```
Epoch 9/20
42/42 [==============================] - 322s 7s/step - loss: 0.2978 - accuracy: 0.8780 - val_loss: 1.1571 - val_a
ccuracy: 0.4879
Epoch 10/20
42/42 [==============================] - 417s 10s/step - loss: 0.2440 - accuracy: 0.9063 - val_loss: 1.3992 - val_
accuracy: 0.4879
Epoch 11/20
42/42 [==============================] - 325s 8s/step - loss: 0.2666 - accuracy: 0.9025 - val_loss: 1.6961 - val_a
ccuracy: 0.4879
Epoch 12/20
42/42 [==============================] - 452s 11s/step - loss: 0.2569 - accuracy: 0.8952 - val_loss: 2.0092 - val_
accuracy: 0.4879
Epoch 13/20
42/42 [==============================] - 327s 8s/step - loss: 0.2372 - accuracy: 0.9151 - val_loss: 2.2223 - val_a
ccuracy: 0.4879
Epoch 14/20
42/42 [==============================] - 388s 9s/step - loss: 0.2390 - accuracy: 0.9057 - val_loss: 2.2535 - val_a
ccuracy: 0.4879
Epoch 15/20
42/42 [==============================] - 326s 8s/step - loss: 0.2495 - accuracy: 0.9092 - val_loss: 1.9416 - val_a
ccuracy: 0.4879
Epoch 16/20
42/42 [==============================] - 323s 7s/step - loss: 0.2640 - accuracy: 0.8789 - val_loss: 1.3799 - val_a
ccuracy: 0.4879
Epoch 17/20
42/42 [==============================] - 327s 8s/step - loss: 0.2270 - accuracy: 0.9166 - val_loss: 0.8044 - val_a
ccuracy: 0.5091
Epoch 18/20
42/42 [==============================] - 340s 8s/step - loss: 0.2235 - accuracy: 0.9124 - val_loss: 0.4153 - val_a
ccuracy: 0.7788
Epoch 19/20
42/42 [==============================] - 336s 8s/step - loss: 0.2272 - accuracy: 0.9102 - val_loss: 0.2623 - val_a
ccuracy: 0.9121
Epoch 20/20
42/42 [==============================] - 382s 9s/step - loss: 0.2384 - accuracy: 0.9064 - val_loss: 0.2130 - val_a
ccuracy: 0.9182
```

Fig. 13. Entraînement du modèle en utilisant SGD comme algorithme d'optimisation avec un learning rate de 0.001

3.2.1.3 VISUALISATION DE RÉSULTATS

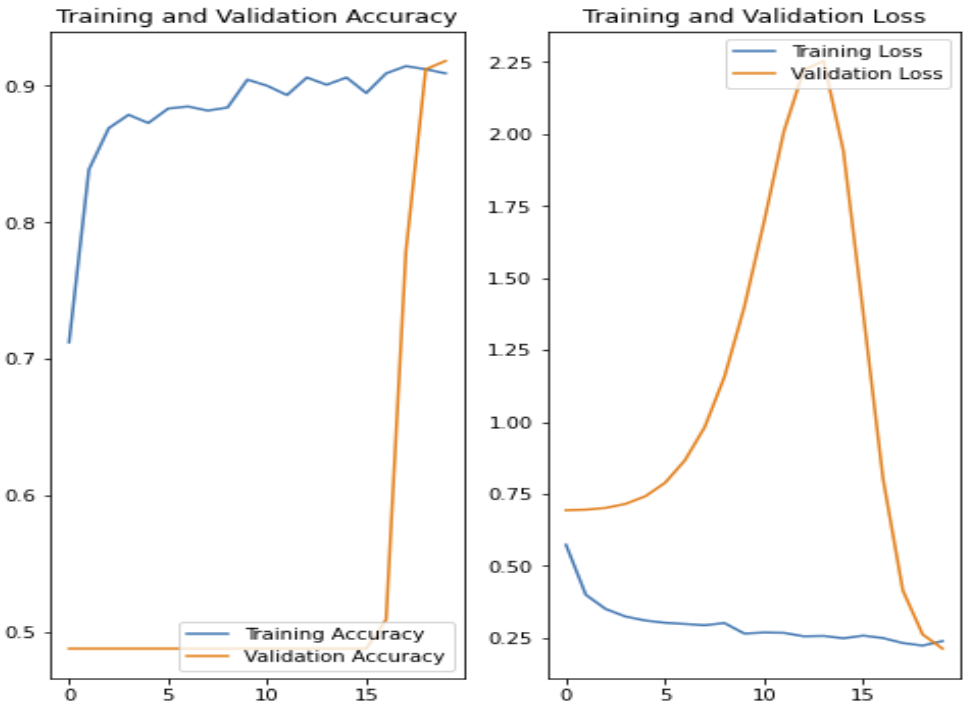

Fig. 14. Visualisation des résultats du modèle en utilisant SGD comme algorithme d'optimisation avec un learning rate de 0.001

3.3 TABLEAU COMPARATIF DES RÉSULTATS

Tableau 1. Tableau comparatif des résultats

Optimizer	Xception			Nbre epoch	Learning rate	Train accuracy	Validation accuracy	Train loss	Validation loss	Image size	Batch size
	1er couche	2 Couches	2e couche								
Adam	C-cov		C-cov								
	32		64	30	0.001	93	75	25	100	180	32
	Stride		Stride								
	2		2	30	0.01	91	76	22	200	180	32
	Padding		Padding								
	same		same	30	0.5	65	52	60	70	180	32
Sgd	C-cov		C-cov								
	32		64								
	Stride		Stride								
	2		2								
	Padding		Padding								
	same		same	20	0.001	91	92	23	21	180	32

4 DISCUSSION

Au point 3.1.1, nous avons commencés avec l'algorithme d'optimisation Adam et un learning rate de 0.001. Après l'entraînement nous avons constaté que notre modèle n'apprend pas bien sur les données. La précision de ce modèle aux données d'entraînement est bonne mais quand nous le mettons face aux données de validation notre modèle fait des mauvaises prédictions (voir la figure 5). Or dans l'apprentissage machine, le plus important est la capacité qu'a le modèle à généraliser, c'est-à-dire à faire des bonnes prédictions sur des nouvelles données. Ce qui n'est pas le cas pour ce modèle. Comme conclusion, nous ne pouvons pas retenir un modèle avec cette valeur de learning rate, parce qu'il fera des mauvaises prédictions sur les données de validation.

Au point 3.1.2, nous avons opté pour une une nouvelle valeur de learning rate, celle-ci est de 0.01. Pour cette valeur de learning rate, nous avons constaté que notre réseau de neurones n'apprend pas. La précision sur les données d'entraînement est bonne mais sa précision sur les données de validation n'est pas bonne (voir la figue 8). Comme conclusion, nous pouvons dire qu'il est difficile de retenir ce modèle pour notre système parce qu'il a un taux d'erreur très élevés sur les données.

Après avoir essayé l'algorithme d'optimisation Adam avec un learning rate de 0.001 et de 0.01, nous allons maintenant essayer avec un learning rate de 0.5. Après avoir entraîné notre modèle, nous avons pu constater que le modèle n'apprend toujours pas bien sur nos données, cette fois la précision sur les données d'entraînement et sur les données de validation a considérablement diminuer au lieu d'augmenter (voir la figure 11).

Nous savons que pour valider un modèle nous devons ajuster les hyperparamètres de ce modèle pour pouvoir minimiser l'erreur d'apprentissage ou la fonction de perte. Or dans notre cas, quand nous avons utilisé l'algorithme d'optimisation Adam, nous n'avons pas pu minimiser la fonction de perte de notre modèle. Nous allons à présent essayer un autre algorithme d'optimisation qui est le Sgd.

Notons que notre réseau de neurones artificiel atteint des bonnes valeurs de prédiction lorsqu'il est aux environs de 19 et 20 epoch, après ces deux valeurs la qualité de prédiction a tendance à rester constante, voilà pourquoi pour cet algorithme d'optimisation nous avons utilisé 20 epochs.

Pour l'algorithme d'optimisation SGD, nous avons utilisé un learning rate (taux d'apprentissage) de 0.001. Après l'entraînement de notre modèle, nous avons pu constater que la précision (accuracy) sur les données d'entraînement est bonne, elle est de 91 %. La précision sur les données de validation est de 92 %. Le taux d'erreur, soit l'erreur d'entraînement tout comme

pour l'erreur de validation est aussi minimiser (voir la figure 14). Vu la taille de notre dataset et la puissance de calcul que nous disposons actuellement, nous ne sommes pas à mesure d'améliorer la précision de notre modèle sur les données de validation en passant de 92 % à 97 ou 98 %.

Nous pouvons conclure en disant qu'en utilisant l'algorithme d'optimisation Sgd et un learning rate (taux d'apprentissage) de 0.001, notre modèle a atteint une bonne capacité de généralisation, parce qu'il arrive à faire des prédictions correctes sur de nouvelles données, qui n'ont pas été utilisées pour le construire.

5 CONCLUSION

Au terme de notre travail, nous avons dans un premier temps examiner les effets néfastes de la pneumonie sur la santé des enfants de moins de cinq ans dans le monde entier, puis nous nous sommes concentrés sur la situation de cette maladie en République Démocratique du Congo en nous appuyant sur des rapports et analyses effectués par des organismes nationaux et internationaux.

Pour concevoir ce système de détection de la pneumonie, nous avons utilisé une approche basée sur l'intelligence artificielle en utilisant les réseaux de neurone convolutif. Étant donné que les réseaux de neurones convolutifs ont plusieurs variantes, nous avons utilisé dans le cadre de notre travail un réseau convolutif appelé « Xception » qui est un réseau convolutif développé par Google. Ce réseau remplace les modules de création par un type de couche appelé convolution séparable en profondeur, alors qu'une couche convolutive régulière utilise des filtres qui tentent de capturer simultanément des motifs spatiaux et modèles de canaux, une couche convolutive séparable fait l'hypothèse forte que les modèles spatiaux et les modèles transcanaux peuvent être modélisés séparément.

Étant donné que l'approche utilisé demande une grande quantité des données pour bien fonctionner, nous étions dans l'obligation de faire appel à un jeu des données (dataset) ayant une grande quantité des données pour entraîner notre système à bien distinguer les clichés pathologies et non pathologies, voilà pourquoi nous avons utilisé le jeu des données du plate-forme web Kaggle ayant 5 863 images radiographiques thoraciques. Nous avons utilisé le langage de programmation Python et ses librairies Numpy, Numpy, Matplotlib, TensorFlow et Keras pour implémenter notre système.

Pour tout système basé sur l'apprentissage machine, qui est une sous branche de l'intelligence artificielle, il nous faut faire la validation des résultats et régler certains hyperparamètres pour obtenir le meilleur résultat. Pour ce faire, nous avons utilisé deux algorithmes d'optimisation « Adam » et « Sgd » avec différentes valeurs du learning rate. Après analyse des différents résultats obtenus sur base de ces algorithmes d'optimisation, nous avons opté pour l'algorithme d'optimisation « Sgd » avec un taux d'apprentissage de 0.001 pour 20 epochs. Avec ces valeurs, notre système a une précision de 92 % sur les nouvelles données.

REMERCIEMENTS

Nos remerciements à l'Institut Supérieur de Techniques Appliquées de Kinshasa (ISTA-Kin) pour le soutient et l'accompagnement dans nos recherches.

REFERENCES

[1] Analyse du paysage du Marché de la pneumonie en République Démocratique du Congo (RDC), USA, 2018, p.21
[2] Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, canada, O'Reilly Media, p.368, 2017.
[3] CHOLLET, François, Xception: Deep Learning with Depthwise Separable Convolutions, 2016
[4] Christopher M. Bishop, Pattern Recognition and Machine Learning, Californie, springer, 2006, p.240
[5] Dimitri P. Bertsekas, Convex Optimization Algorithms, Usa, Athena Scientific, 2015, p.53.
[6] Ian Goodfellow et all, Deep Learning, 2016, p.240
[7] Jérémy WATT et all, Machine Learning Refined Foundations, Algorithms, and Applications, New York, Cambridge press, p.229, 2016
[8] M.GRAWITZ, Méthode en sciences sociales, Paris, Dalloz 1986, p.15
[9] Nikhil Buduma and Nicholas Lacascio, Fundamentals of Deep Learning Designing Next-Generation Machine Intelligence Algorithms, Usa, 2017, p.81
[10] SIMON Jean-Jacques, https://www.unicef.org/drcongo/communiques-de-presse/pneumonie-plusieurs-organismes-tirent-sonnette-alarme
[11] https://www.who.int/fr/news-room/fact-sheets/detail/pneumonia
[12] https://www.who.int/fr/news-room/fact-sheets/detail/pneumonia.
[13] http://www.kaggle.org