MODÉLISATION ET RÉSEAUX DE NEURONES FORMELS

Anthony PEDROSA DOS SANTOS

Réseau de neurones artificiel : « Breast Cancer Prediction Dataset »

Explication théorique : le réseau de neurone artificiel

Pour traiter ces données et espérer prédire un cancer du sein, le *deep learning* apparait comme un outil adapté. Le *deep learning* fait partie du domaine de l'intelligence artificielle. C'est une technique adaptée pour émettre des prédictions à partir d'un jeu de données. Pour faire des prédictions correctes, la machine va devoir apprendre. Un bon moyen d'y parvenir est de développer un réseau de neurones artificiel.

Un réseau de neurone est composé d'une succession de couches dont chacune est composée d'un certain nombre de neurones. La première couche de neurones est appelée « couche d'entrée ». Les autres couches de neurones sont appelées « couches cachées ». La dernière couche de neurones est appelée « couche de sortie ». Tous les neurones sont connectés entre eux par des liens qui vont leur permettre une transmission de l'information. Pour chaque neurone, il est associé un poids synaptique qui va modifier la valeur reçue. Cette valeur est modifiée par le neurone à l'aide de la fonction d'activation. Celle-ci va permettre au neurone de déterminer si la valeur doit être transmise ou non au(x) neurone(s) suivant(s). Si la valeur est proche de 1, elle sera transmise, et si elle est proche de 0, elle ne le sera pas. C'est donc une analogie à ce que l'on retrouve chez le cerveau humain, où certains neurones s'activent et d'autres non, en fonction de l'information reçue.

Concrètement, dans un premier temps, je vais donner des valeurs aux neurones de la « couche d'entrée » qui correspondent aux données que je souhaite traiter. Ces valeurs vont ensuite être modifiées par les poids synaptiques et certaines seront transmises aux neurones suivants. Ce procédé va se répéter couches de neurones après couches de neurones jusqu'au(x) neurone(s) de la « couche de sortie ». Ce(s) neurone(s) de la « couche de sortie » va(vont) modifier les valeurs reçues et donnera une prédiction. Cependant, cette prédiction a peu de chance d'être exacte puisque mon réseau n'a pas encore été entrainé. En effet, les poids initiaux sont

aléatoirement assignés. Le(s) neurone(s) de la « couche de sortie » va prendre en considération la valeur réelle attendue et pourra, à l'aide d'une fonction de coût, modifier les poids synaptiques du réseau pour obtenir une valeur en sortie proche de la valeur réelle attendue selon une certaine méthode appelée « méthode de rétropropagation ».

Détail du sujet d'étude

J'ai choisi d'utiliser des données permettant de prédire si une femme est atteinte ou non d'un cancer du sein à partir de certaines caractéristiques.

Ces données ont été téléchargées sur le site *Kaggle*. Elles sont disponibles sur ce lien : https://www.kaggle.com/merishnasuwal/breast-cancer-prediction-dataset

Ce jeu de donnée a été obtenue auprès des hôpitaux de l'université du Wisconsin par le Dr. William H. Wolberg. Lorsqu'il trouvait une bosse anormale à partir d'une examinassions médicale, un diagnostique pour le cancer du sein était engagé. Il récoltait plusieurs données à partir de cette bosse suspicieuse, telle que le rayon moyen, la texture moyenne, le périmètre moyen, la surface moyenne et la régularité moyenne. Il détaillait ensuite si la patiente était atteinte d'un cancer du sein ou non. Ainsi, toutes les données de toutes les colonnes étaient pertinentes. Je n'ai pas eu besoin de supprimer certaines données.

Les données détaillant les caractéristiques de la bosse suspicieuse étaient toutes des nombres d'échelles différentes. J'ai donc été amené à les normaliser. En effet, la normalisation est appliquée aux données lorsqu'elles sont d'échelles différentes. Cela permet d'éviter certaines erreurs (par exemple, comparer des miles et des kilomètres). La normalisation permet alors de transformer les données pour les convertir et obtenir une échelle commune entre les données. Concernant les données stockées en sorties, elles étaient binaires : 1 correspondait à la présence d'un cancer du sein et 0 à la non-présence d'un cancer du sein. Il y avait légèrement plus de données issues de patientes atteintes d'un cancer du sein (62.74%) que de patientes saines (37.26%). Néanmoins, le jeu de données n'était pas déséquilibré au point d'affecter les prédictions. Je n'ai donc pas été amené à sous-échantillonner ou sur-échantillonner les

données. Ce constat sera illustré par une matrice de confusion dans une prochaine partie de ce devoir.

Enfin, j'ai divisé les données afin d'avoir 80% d'entre-elles dédiées à l'entrainement du réseau et 20% dédiés au test.

L'architecture du réseau utilisé

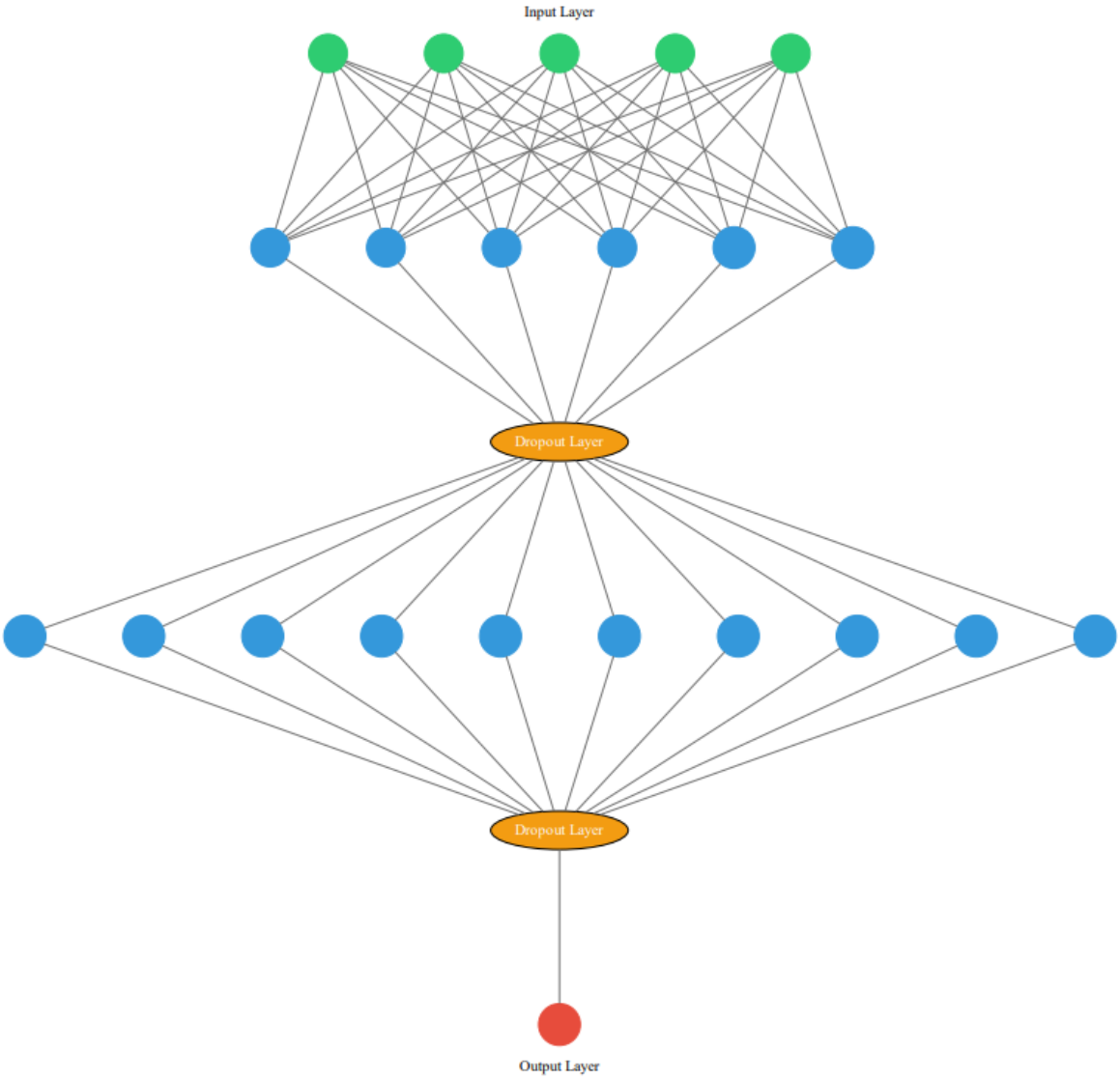


Figure 1 : Visualisation du réseau de neurones artificiel.

Dans la mesure où le jeu de données fournissait 5 types de données concernant les caractéristiques des bosses suspicieuses, le réseau de neurone contient 5 neurones dans sa couche d'entrée. Ces valeurs vont ensuite passer par deux couches de neurones cachés. Etant donné que la prédiction attendue est binaire (1 si la patiente est atteinte d'un cancer du sein, 0 si elle est saine), un seul neurone est présent dans la couche de sortie (*cf. Figure 1*). Les détails des hyperparamètres utilisés seront détaillés dans la partie suivante.

Les différents essais et les hyperparamètres utilisés

Pour entrainer ce réseau de neurones artificiel, j'ai utilisé la classe GridSearchCV() fournie par le module *Scikit-Learn.* Cette dernière m'a permis de trouver les hyperparamètres adaptés pour obtenir la meilleure précision possible.

Dans un premier temps, j'ai modifié les valeurs de la classe Dropout() fournie par le module Keras afin d'éviter le surapprentissage, tout en essayant de ne pas obtenir une prédiction trop faible. Celle-ci permet de briser aléatoirement les liens entre deux neurones afin d'éviter que ces liens ne soient trop forts et mènent vers un surapprentissage du réseau. Le paramètre entré dans cette classe correspond à la chance qu'a le lien entre les couches d'être brisé (par exemple, 0.1 donne lieu à une chance de 10% que le lien entre les neurones soit brisé lors d'une époque). J'ai testé les valeurs 0.1 (*cf. Figure 2*), 0.2 (*cf. Figure 3*) et 0.4 (*cf. Figure 4*) de façon totalement aléatoire, et après plusieurs essais, j'ai conservé la valeur 0.2 qui permettait au réseau d'éviter correctement le surapprentissage (*cf. Figure 9 – Matrice de Confusion*) tout en conservant une prédiction élevée (la meilleure prédiction fournie par l'attribut « best_score_ » de la classe GridSearchCV() oscillait entre 92.5% et 93.14%). J'ai aussi tenté d'ajouter d'autres couches de neurones, mais cela avait pour effet de réduire la précision.

```
38
39  #2 - CONSTRUCTION DU RÉSEAU DE NEURONES OPTIMISE
40  from keras.layers import Dense
41  from keras.layers import Dropout
42  from keras.wrappers.scikit_learn import KerasClassifier
43  from sklearn.model_selection import GridSearchCV
44
45  def create_network_opt(algo_gradient,nb1,nb2):
46      network = Sequential()
47      network.add(Dense(units = nb1, kernel_initializer = 'uniform', activation = 'relu', input_dim = 5))
48      network.add(Dropout(rate=0.1))
49      network.add(Dense(units = nb2, kernel_initializer = 'uniform', activation = 'relu'))
50      network.add(Dropout(rate=0.1))
51      network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
52      network.compile(optimizer = algo_gradient, loss = 'binary_crossentropy', metrics=['accuracy'])
53      return network
54
55  networkOpt=KerasClassifier(build_fn=create_network_opt)
56  hyperparam={'batch_size':[10,25],'epochs':[10,50],'algo_gradient':['adam','rmsprop'],'nb1':[6,10],'nb2':[6,10]}
57  grid_search=GridSearchCV(estimator=networkOpt, param_grid=hyperparam,cv=10)
58  grid_search=grid_search.fit(X,y)
59
60  best_precision=grid_search.best_score_
61  best_params=grid_search.best_params_
62
```

Figure 2 : Test 1/4 des hyperparamètres du réseau de neurones artificiel.

```
38
39  #2 - CONSTRUCTION DU RÉSEAU DE NEURONES OPTIMISE
40  from keras.layers import Dense
41  from keras.layers import Dropout
42  from keras.wrappers.scikit_learn import KerasClassifier
43  from sklearn.model_selection import GridSearchCV
44
45  def create_network_opt(algo_gradient,nb1,nb2):
46      network = Sequential()
47      network.add(Dense(units = nb1, kernel_initializer = 'uniform', activation = 'relu', input_dim = 5))
48      network.add(Dropout(rate=0.4))
49      network.add(Dense(units = nb2, kernel_initializer = 'uniform', activation = 'relu'))
50      network.add(Dropout(rate=0.4))
51      network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
52      network.compile(optimizer = algo_gradient, loss = 'binary_crossentropy', metrics=['accuracy'])
53      return network
54
55  networkOpt=KerasClassifier(build_fn=create_network_opt)
56  hyperparam={'batch_size':[10,25],'epochs':[10,50],'algo_gradient':['adam','rmsprop'],'nb1':[6,10],'nb2':[6,10]}
57  grid_search=GridSearchCV(estimator=networkOpt, param_grid=hyperparam,cv=10)
58  grid_search=grid_search.fit(X,y)
59
60  best_precision=grid_search.best_score_
61  best_params=grid_search.best_params_
62
63
64  #3 - APPLICATION DES MEILLEURS PARAMETRES TROUVES POUR OBTENIR LA VARIANCE ET LA MOYENNE
```

Figure 3 : Test 2/4 des hyperparamètres du réseau de neurones artificiel.

```
38
39      #2 - CONSTRUCTION DU RÉSEAU DE NEURONES OPTIMISE
40      from keras.layers import Dense
41      from keras.layers import Dropout
42      from keras.wrappers.scikit_learn import KerasClassifier
43      from sklearn.model_selection import GridSearchCV
44
45      def create_network_opt(algo_gradient,nb1,nb2):
46          network = Sequential()
47          network.add(Dense(units = nb1, kernel_initializer = 'uniform', activation = 'relu', input_dim = 5))
48          network.add(Dropout(rate=0.2))
49          network.add(Dense(units = nb2, kernel_initializer = 'uniform', activation = 'relu'))
50          network.add(Dropout(rate=0.2))
51          network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
52          network.compile(optimizer = algo_gradient, loss = 'binary_crossentropy', metrics=['accuracy'])
53          return network
54
55      networkOpt=KerasClassifier(build_fn=create_network_opt)
56      hyperparam={'batch_size':[10,25],'epochs':[10,50],'algo_gradient':['adam','rmsprop'],'nb1':[6,10],'nb2':[6,10]}
57      grid_search=GridSearchCV(estimator=networkOpt, param_grid=hyperparam,cv=10)
58      grid_search=grid_search.fit(X,y)
59
60      best_precision=grid_search.best_score_
61      best_params=grid_search.best_params_
62
63
```

Figure 4 : Test 3/4 des hyperparamètres du réseau de neurones artificiel.

Enfin, j'ai ajouté des hyperparamètres à tester par GridSearchCV() tel que le nombre d'époques, les algorithmes du gradient (SGD et Adagrad) et le nombre de neurones dans les couches cachées (*cf. Figure 5*).

```
38
39      #2 - CONSTRUCTION DU RÉSEAU DE NEURONES OPTIMISE
40      from keras.layers import Dense
41      from keras.layers import Dropout
42      from keras.wrappers.scikit_learn import KerasClassifier
43      from sklearn.model_selection import GridSearchCV
44
45      def create_network_opt(algo_gradient,nb1,nb2):
46          network = Sequential()
47          network.add(Dense(units = nb1, kernel_initializer = 'uniform', activation = 'relu', input_dim = 5))
48          network.add(Dropout(rate=0.2))
49          network.add(Dense(units = nb2, kernel_initializer = 'uniform', activation = 'relu'))
50          network.add(Dropout(rate=0.2))
51          network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
52          network.compile(optimizer = algo_gradient, loss = 'binary_crossentropy', metrics=['accuracy'])
53          return network
54
55      networkOpt=KerasClassifier(build_fn=create_network_opt)
56      hyperparam={'batch_size':[10,25],'epochs':[10,25,50],'algo_gradient':['adam','rmsprop', 'sgd', 'adagrad'],'nb1':[6,8,10,12,14],'nb2':[6,8,10,12,14]}
57      grid_search=GridSearchCV(estimator=networkOpt, param_grid=hyperparam,cv=10)
58      grid_search=grid_search.fit(X,y)
59
60      best_precision=grid_search.best_score_
61      best_params=grid_search.best_params_
62
63
```

Figure 5 : Test 4/4 des hyperparamètres du réseau de neurones artificiel.

Au final, après plusieurs essais, il apparait que les meilleurs paramètres sont ceux affichés dans la Figure 6, obtenues grâce à l'attribut « best_params_ » de la classe GridSearchCV(). Concrètement, Ces hyperparamètres fournissent une précision la plus haute de **93.14%** (*cf. Figure 7*) tout en conservant une faible variance (σ=0.04) (*cf. Figure 8*). Le modèle entrainé permet d'émettre de bonnes prédictions, notamment grâce à un jeu de donnée plutôt équilibré

(Il y avait légèrement plus de données issues de patientes atteintes d'un cancer du sein (62.74%) que de patientes saines (37.26%)). Ce constat est confirmé par la matrice de confusion obtenue qui montre que le modèle peut prédire des valeurs de sortie significativement différentes (concrètement : des valeurs proches de 0 et des valeurs proches de 1) (*cf. Figure 9*).

La fonction de coût « binary-crossentropy » a été choisie car il n'existe qu'un seul neurone dans la couche de sortie. La fonction d'activation pour la couche de sortie est la fonction sigmoïde car il n'existe qu'un seul neurone dans la couche de sortie.

Key	Type	Size	
algo_gradient	str	1	adam
batch_size	int	1	10
epochs	int	1	50
nb1	int	1	6
nb2	int	1	10

Figure 6 : Meilleurs hyperparamètres obtenus par la classe GridSearchCV().

best_precision	float64	1	0.9313909888267518

Figure 7 : Meilleures précision obtenue par la classe GridSearchCV().

ecart_type	float64	1	0.04105565887270473

Figure 8 : Variance obtenue par le modèle.

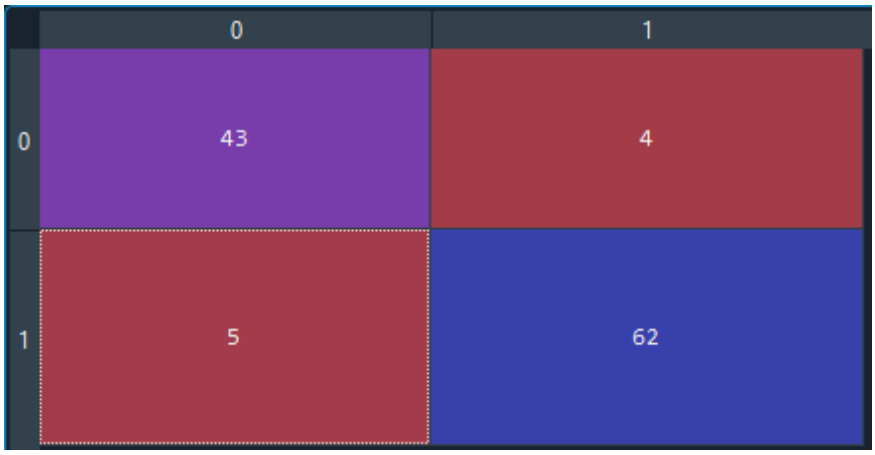


Figure 9 : Matrice de confusion du modèle.

Réseau de neurones convolutif : « Pneumonia X-Ray Images »

Explication théorique : le réseau de neurones convolutif

Pour pouvoir traiter un ensemble d'images afin d'émettre des prédictions, je vais développer un réseau de neurones convolutif. Le principe d'un réseau neuronal convolutif est de filtrer les images à traiter avant d'entrainer le réseau de neurones. Après avoir filtré une image, les caractéristiques de celles-ci seront plus faciles à traiter pour émettre des prédictions.

Un filtre est simplement un ensemble de multiplicateur. Lorsque le réseau va se focaliser sur un pixel spécifique, il va prendre en compte les valeurs numériques de chaque couleur primaire (rouge, bleu et vert) et va les multiplier avec les valeurs respectives du filtre. La nouvelle valeur du pixel sera la somme de toutes ces multiplications.

Cette étape peut être combinée avec ce que l'on appelle « la couche de pooling ». C'est une forme de sous-échantillonnage de l'image qui permet le regroupement des pixels de l'image et les filtres en un sous-ensemble. Ainsi, si on décide de faire subir une étape de pooling à l'image selon un processus de Max-Pooling de 2x2, les pixels de l'image seront rassemblés en ensembles de 2x2, et seul le pixel avec la plus haute valeur d'origine sera retenue. L'image sera donc réduite au quart de sa taille d'origine, mais ses caractéristiques seront conservées.

Tout comme pour les poids synaptiques des réseaux de neurones artificiels, les filtres sont aléatoirement assignés initialement. C'est suite à l'entrainement du neurone que les filtres seront, au fur et à mesure, modifiés.

Ainsi, en obtenant des valeurs purement numériques à partir d'une image, le réseau neuronal peut s'entrainer et assimiler les caractéristiques de plusieurs images et les assigner à certaines catégories. C'est grâce aux filtres et aux étapes de pooling que ce processus est, dans une certaine mesure, rapide.

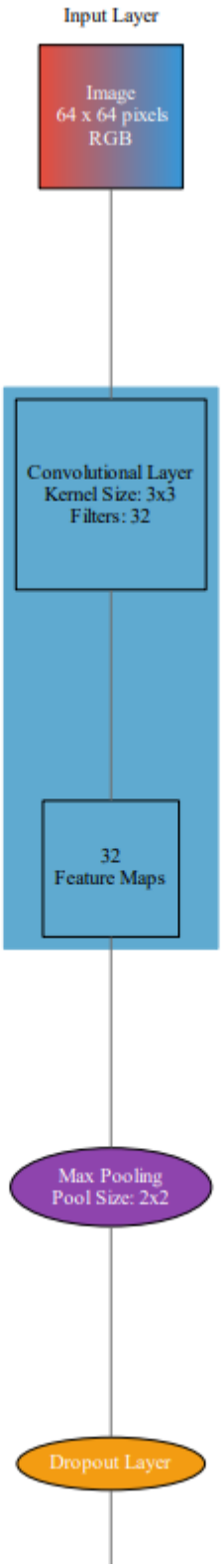
Détail du sujet d'études

J'ai choisi d'utiliser des images de poumons pour prédire si un poumon est sain ou affecté par une pneumonie.

Ces images ont été téléchargées sur Kaggle. Elles sont disponibles sur le lien : https://www.kaggle.com/pcbreviglieri/pneumonia-xray-images

Elles proviennent d'un autre jeu de données où il existait un déséquilibre trop important entre le nombre de poumons sains et le nombre de poumons affectés par une pneumonie. Ici, la proportion est plutôt équilibrée (27.33% poumons sains VS 72.67% poumons affectés), cela permettra d'émettre des prédictions fiables pour les deux cas (sains VS affectés). Les données étaient déjà divisées afin d'obtenir une partie dédiée à la phase d'entrainement et une autre à la phase de test. Dans la mesure où 87% des données étaient dédiées à la phase d'entrainement et 13% seulement à la phase de test, il est possible de craindre un surapprentissage. J'ai donc pris mes précautions pour éviter ce phénomène, qui seront détaillées plus tard dans ce devoir.

L’architecture du réseau utilisé



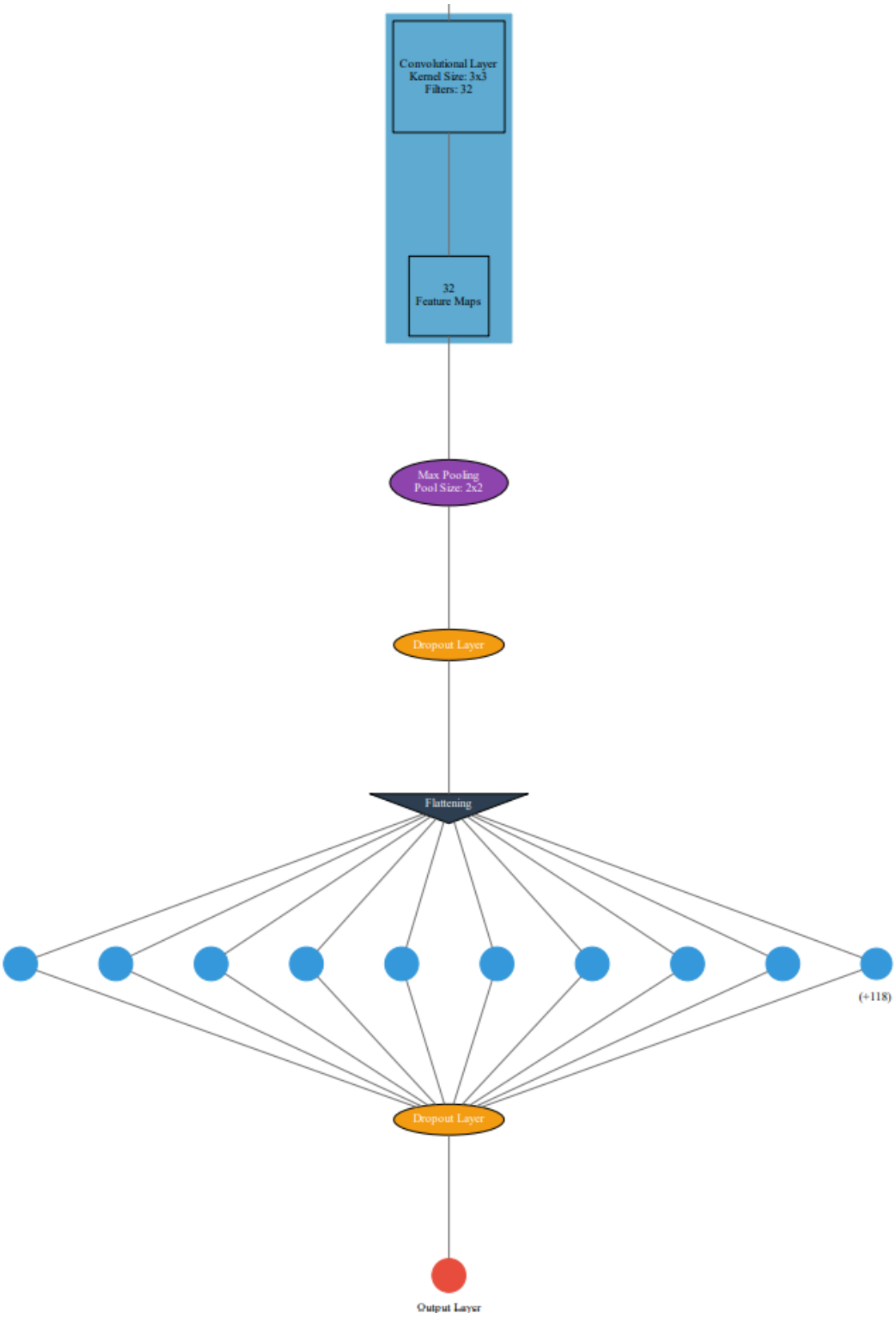


Figure 10 : Architecture du réseau de neurones convolutif.

Les images ont été redimensionnées à 64x64 pour augmenter la vitesse de l'entrainement. La couche d'entrée contient 128 neurones pour augmenter la vitesse de l'entrainement tout en conservant une bonne précision. Etant donné que la prédiction attendue est binaire (1 si le poumon est affecté par une pneumonie, 0 s'il est sain), un seul neurone est présent dans la couche de sortie (*cf. Figure 10*). Les détails des hyperparamètres utilisés seront détaillés dans la partie suivante.

Les différents essais et les hyperparamètres utilisés

Au départ j'ai opté pour une seule étape de convolution (taille du kernel 3x3) et une seule étape de pooling (cf. Figure 11). Je me suis rapidement aperçu que je pouvais obtenir une meilleure précision (96.12%) avec quelques modifications.

J'ai ensuite ajouté 2 étapes de convolution et de pooling, tout en gardant les mêmes caractéristiques que les premières étapes : une taille du kernel 3x3, 32 filtres et un déplacement (stride) de 1 pixel par 1 pixel (cf. Figure 12). Mais j'ai obtenu une précision plus mauvaise que précédemment (92.45%).

J'ai donc tenté de modifier le nombre de filtres pour chaque étape de convolution. D'abord en mettant respectivement 16, 32 et 64 filtres pour chaque étape de convolution (cf. Figure 13) et ensuite en mettant respectivement 64, 32 et 16 filtres pour chaque étape de convolution (cf. Figure 14). Malheureusement, ça a diminué la prédiction obtenue dans chacun des tests (88.32% pour le premier, 90.55% pour le deuxième).

J'ai donc remis la taille des filtres à 32, comme initialement. Mais j'ai ajouté des étapes de dropout grâce à la classe Dropout() du module *Keras*. En effet, comme précisé précédemment, il y a un certain déséquilibre entre le nombre d'images dédiées à la phase d'entrainement et le nombre d'images dédiées à la phase de test. Cela peut avoir pour effet de provoquer un

surapprentissage du réseau. Des étapes de dropout permettent de casser aléatoirement certains liens entre les neurones pour éviter ce phénomène de surapprentissage. J'ai initialement testé de casser ces liens dans 10% des cas pour chaque couche de dropout (cf. Figure 15) ce qui m'a permis d'obtenir une bonne précision (90.08%) puis ait testé d'autres valeurs plus hautes (cf. Figure 16) avant de me raviser suite à la diminution de la prédiction (92.63%).

J'ai ensuite essayé d'ajouter simplement une étape de convolution supplémentaire après chaque étape de convolution (cf. Figure 17) mais j'ai obtenu une diminution significative de la prédiction (89.82%).

J'ai donc essayé de retiré le troisième bloc composé de deux étapes de convolutions et d'une étape de pooling (cf. Figure 18) sans grand effet sur la prédiction (90.35%).

Au final il apparait que les meilleurs paramètres soient ceux de la Figure 19 : deux blocs identiques composés d'une étape de convolution à 32 filtres avec une taille de kernel à 3x3 et un déplacement (stride) à 1. Suivie d'une étape de pooling de 2x2 pixels. J'ai obtenu au final une prédiction de **97.30%** avec ce modèle.

Enfin, les étapes par époques ont été définies à 131 car il y a 4192 images pour la phase d'entrainement et 32 filtres dans les étapes de convolutions (4192/32=131). De même pour le nombre d'étapes de validation définie à 20 car il y a 624 images pour l'étape de test et 32 filtres dans les étapes de convolution (624/32=19.5).

La fonction de coût « binary-crossentropy » a été choisie car il n'existe qu'un seul neurone dans la couche de sortie. La fonction d'activation pour la couche de sortie est la fonction sigmoïde car il n'existe qu'un seul neurone dans la couche de sortie. J'ai décidé de faire tourner le réseau pendant 50 époques pour ne pas que l'entrainement ne dure trop longtemps tout en gardant une bonne précision.

```
28
29  #CONSTRUCTION ET ENTRAINEMENT DU RESEAU
30  from keras.models import Sequential
31  from keras.layers import Dense
32  from keras.layers import Conv2D
33  from keras.layers import MaxPooling2D
34  from keras.layers import Flatten
35  from keras.layers import Dropout
36
37
38  network = Sequential()
39
40  network.add(Conv2D(filters=32, kernel_size = 3, strides = 1, input_shape = (64,64,3), activation = 'relu'))
41  network.add(MaxPooling2D(pool_size=(2,2)))
42
43  network.add(Flatten())
44  network.add(Dense(units = 128, kernel_initializer = 'uniform', activation = 'relu'))
45  network.add(Dropout(rate=0.1))
46
47  network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
48  network.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics=['accuracy'])
49
50  network.fit(
51          train_generator,
52          steps_per_epoch=131,
53          epochs=50,
54          validation_data=validation_generator,
55          validation_steps=20)
56
57  print("Terminé")
58
```

Figure 11 : Test 1/9 des hyperparamètres du réseau de neurones convolutif.

```
28
29  #CONSTRUCTION ET ENTRAINEMENT DU RESEAU
30  from keras.models import Sequential
31  from keras.layers import Dense
32  from keras.layers import Conv2D
33  from keras.layers import MaxPooling2D
34  from keras.layers import Flatten
35  from keras.layers import Dropout
36
37
38  network = Sequential()
39
40  network.add(Conv2D(filters=32, kernel_size = 3, strides = 1, input_shape = (64,64,3), activation = 'relu'))
41  network.add(MaxPooling2D(pool_size=(2,2)))
42
43  network.add(Conv2D(filters=32, kernel_size = 3, strides = 1,  activation = 'relu'))
44  network.add(MaxPooling2D(pool_size=(2,2)))
45
46
47  network.add(Conv2D(filters=32, kernel_size = 3, strides = 1, activation = 'relu'))
48  network.add(MaxPooling2D(pool_size=(2,2)))
49
50  network.add(Flatten())
51  network.add(Dense(units = 128, kernel_initializer = 'uniform', activation = 'relu'))
52  network.add(Dropout(rate=0.1))
53
54  network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
55  network.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics=['accuracy'])
56
57  network.fit(
58          train_generator,
59          steps_per_epoch=131,
60          epochs=50,
61          validation_data=validation_generator,
62          validation_steps=20)
```

Figure 12 : Test 2/9 des hyperparamètres du réseau de neurones convolutif.

```
#CONSTRUCTION ET ENTRAINEMENT DU RESEAU
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dropout


network = Sequential()

network.add(Conv2D(filters=16, kernel_size = 3, strides = 1, input_shape = (64,64,3), activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))

network.add(Conv2D(filters=32, kernel_size = 3, strides = 1,  activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))


network.add(Conv2D(filters=64, kernel_size = 3, strides = 1, activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))

network.add(Flatten())
network.add(Dense(units = 128, kernel_initializer = 'uniform', activation = 'relu'))
network.add(Dropout(rate=0.1))

network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
network.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics=['accuracy'])

network.fit(
        train_generator,
        steps_per_epoch=131,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=20)
```

Figure 13 : Test 3/9 des hyperparamètres du réseau de neurones convolutif.

```
28
29  #CONSTRUCTION ET ENTRAINEMENT DU RESEAU
30  from keras.models import Sequential
31  from keras.layers import Dense
32  from keras.layers import Conv2D
33  from keras.layers import MaxPooling2D
34  from keras.layers import Flatten
35  from keras.layers import Dropout
36
37
38  network = Sequential()
39
40  network.add(Conv2D(filters=64, kernel_size = 3, strides = 1, input_shape = (64,64,3), activation = 'relu'))
41  network.add(MaxPooling2D(pool_size=(2,2)))
42
43  network.add(Conv2D(filters=32, kernel_size = 3, strides = 1,  activation = 'relu'))
44  network.add(MaxPooling2D(pool_size=(2,2)))
45
46
47  network.add(Conv2D(filters=16, kernel_size = 3, strides = 1, activation = 'relu'))
48  network.add(MaxPooling2D(pool_size=(2,2)))
49
50  network.add(Flatten())
51  network.add(Dense(units = 128, kernel_initializer = 'uniform', activation = 'relu'))
52  network.add(Dropout(rate=0.1))
53
54  network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
55  network.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics=['accuracy'])
56
57  network.fit(
58          train_generator,
59          steps_per_epoch=131,
60          epochs=50,
61          validation_data=validation_generator,
62          validation_steps=20)
```

Figure 14 : Test 4/9 des hyperparamètres du réseau de neurones convolutif.

```
#CONSTRUCTION ET ENTRAINEMENT DU RESEAU
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dropout


network = Sequential()

network.add(Conv2D(filters=32, kernel_size = 3, strides = 1, input_shape = (64,64,3), activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(rate=0.1))

network.add(Conv2D(filters=32, kernel_size = 3, strides = 1,  activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(rate=0.1))


network.add(Conv2D(filters=32, kernel_size = 3, strides = 1, activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(rate=0.1))

network.add(Flatten())
network.add(Dense(units = 128, kernel_initializer = 'uniform', activation = 'relu'))
network.add(Dropout(rate=0.1))

network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
network.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics=['accuracy'])

network.fit(
        train_generator,
        steps_per_epoch=131,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=20)
```

Figure 15 : Test 6/9 des hyperparamètres du réseau de neurones convolutif.

```
#CONSTRUCTION ET ENTRAINEMENT DU RESEAU
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dropout


network = Sequential()

network.add(Conv2D(filters=32, kernel_size = 3, strides = 1, input_shape = (64,64,3), activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(rate=0.1))

network.add(Conv2D(filters=32, kernel_size = 3, strides = 1,  activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(rate=0.3))


network.add(Conv2D(filters=32, kernel_size = 3, strides = 1, activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(rate=0.4))

network.add(Flatten())
network.add(Dense(units = 128, kernel_initializer = 'uniform', activation = 'relu'))
network.add(Dropout(rate=0.1))

network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
network.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics=['accuracy'])

network.fit(
        train_generator,
        steps_per_epoch=131,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=20)
```

Figure 16 : Test 7/9 des hyperparamètres du réseau de neurones convolutif.

```
#CONSTRUCTION ET ENTRAINEMENT DU RESEAU
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dropout


network = Sequential()

network.add(Conv2D(filters=32, kernel_size = 3, strides = 1, input_shape = (64,64,3), activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(rate=0.1))

network.add(Conv2D(filters=32, kernel_size = 3, strides = 1,  activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(rate=0.1))


network.add(Conv2D(filters=32, kernel_size = 3, strides = 1, activation = 'relu'))
network.add(Conv2D(filters=16, kernel_size = 3, strides = 1,  activation = 'relu'))
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(rate=0.1))

network.add(Flatten())
network.add(Dense(units = 128, kernel_initializer = 'uniform', activation = 'relu'))
network.add(Dropout(rate=0.1))

network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
network.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics=['accuracy'])

network.fit(
        train_generator,
        steps_per_epoch=131,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=20)
```

Figure 17 : Test 8/9 des hyperparamètres du réseau de neurones convolutif.

```
27
28
29  #CONSTRUCTION ET ENTRAINEMENT DU RESEAU
30  from keras.models import Sequential
31  from keras.layers import Dense
32  from keras.layers import Conv2D
33  from keras.layers import MaxPooling2D
34  from keras.layers import Flatten
35  from keras.layers import Dropout
36
37
38  network = Sequential()
39
40  network.add(Conv2D(filters=32, kernel_size = 3, strides = 1, input_shape = (64,64,3), activation = 'relu'))
41  network.add(MaxPooling2D(pool_size=(2,2)))
42  network.add(Dropout(rate=0.1))
43
44
45  network.add(Conv2D(filters=32, kernel_size = 3, strides = 1, activation = 'relu'))
46  network.add(MaxPooling2D(pool_size=(2,2)))
47  network.add(Dropout(rate=0.1))
48
49  network.add(Flatten())
50  network.add(Dense(units = 128, kernel_initializer = 'uniform', activation = 'relu'))
51  network.add(Dropout(rate=0.1))
52
53  network.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
54  network.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics=['accuracy'])
55
56  network.fit(
57          train_generator,
58          steps_per_epoch=131,
59          epochs=50,
60          validation_data=validation_generator,
61          validation_steps=20)
62
63  print("Terminé")
64
```

Figure 18 : Test 9/9 des hyperparamètres du réseau de neurones convolutif.

Réflexions sur le machine learning

Réflexion sur les avantages et dérives d'utilisation du machine learning sur les sujets étudiés, sur les biais des algorithmes utilisés, les dérives potentielles et les problèmes d'éthique de la course à l'intelligence artificielle.

J'ai choisi deux jeux de données médicales permettant de prédire un diagnostic. Evidemment, il est pertinent de se questionner quant à la fiabilité de ce diagnostic. Même si on pourrait voir un gros avantage dans ce genre de réseau de neurones (rapidité des résultats, fiabilité hors-norme, accessibilité pour les patients), on peut questionner les ingénieurs derrière l'élaboration de ce réseau. Par exemple, vis-à-vis des données fournies au réseau : est-ce que ces données sont fiables ? Quelles données doit-on donner au réseau ? Qui doit et peut utiliser ce réseau ? Il est aussi pertinent de se demander si le réseau seul peut donner un diagnostique ou si un médecin doit superviser ces résultats. Alors, dans ce cas, le réseau servirait seulement de repère. C'est

pourquoi de nombreuses entreprises investissent massivement dans ce genre de recherche : développer un réseau permettant de diagnostiquer une maladie, grave ou non, avec rapidité et fiabilité et d'une manière plus accessible engrangerait un retour sur investissement énorme. Il est donc tout à fait normal de questionner les nombreuses dérives qui se développent dans le domaine du machine learning à notre époque, car cette technologie permettra à coup sûre d'enrichir de gros investisseur. L'éthique doit être alors placée au premier plan pour éviter tout abus de cette technologie puissante qui peut avoir des effets néfastes sur l'humain (par exemple, mal diagnostiquer une maladie grave).

L'innovation dans ce domaine s'amplifie de jour en jour, et les gouvernements risquent rapidement d'être submergés par de nombreuses dérives (mise sur le marché de réseau non fiables, manque de transparence des réseaux et des données collectées, …) c'est pourquoi non seulement cette pratique doit être encadrée par des lois mais aussi par une certaine éthique établie par les organisations privées elles-mêmes afin de gagner la confiance de ses utilisateurs.

Il existe aussi un problème majeur dans le deep learning : il n'est pas possible de correctement identifier les mécanismes sous-jacents dans les réseaux de neurones. Ces réseaux restent en partie des boites noires, dont on commence à peine à vouloir en maitriser les aspect afin d'obtenir plus de transparence.

Ma vision du futur concernant l'intelligence artificielle

J'ai bonne foi en l'intelligence artificielle pour aider l'humain dans beaucoup de domaine (santé, économie, écologie, développement, recherches scientifiques, cultures, …). Cependant, je pense qu'il ne faut pas avoir honte d'avoir peur de cette technologie. Le fait qu'elle soit très puissante en fait d'elle un outil adapté aux défis de notre époque (réchauffement climatique, pandémies, famines, …) mais les risques qui lui incombent restent nombreux et il faut que les développeurs et utilisateurs restent prudent vis-à-vis de celle-ci. L'exemple du film terminator peut typiquement faire sourire, mais les gouvernements du monde entier investissent massivement dans l'intelligence artificielle au profit de leur puissance militaire. Si l'être humain a tendance lui-même à se considérer comme « mauvais » pour l'environnement (cf. tendance à considérer l'homme comme la maladie et la Covid-19 comme le traitement), pourquoi une machine ne

serais pas sensible aux mêmes biais ? Il faut questionner à la fois la conception, l'utilisation et les objectifs de l'intelligence artificielle afin d'envisager son déploiement massif dans tout les domaines qui nous entourent (santé, éducation, …).