Projet Kaggle : Chest X-Ray (Pneumonia)

Introduction

La pneumonie est une infection des alvéoles et des tissus environnants. La pneumonie est l'une des causes de décès les plus fréquentes au monde. Le diagnostic de cette maladie est souvent réalisé par l'intermédiaire de radiographie du thorax. En effet, l'imagerie médicale est un domaine de plus en plus sollicité dans l'établissement de diagnostic et dans la recherche sur le fonctionnement de l'organisme. Les techniques de visualisation et d'interprétation se renouvellent sans cesse, au travers du développement d'appareils de visualisation de plus en plus performants, proposant des images précises de haute résolution qui facilitent le diagnostic.

Cependant, l'expertise humaine reste limitée, la classification des maladies devient difficile. Le deep learning a révolutionné les méthodes traditionnelles de classification d'images, et pourrait apporter énormément au secteur de l'imagerie médicale. L'intelligence artificielle permettrait aux radiologues d'appuyer leur diagnostic et d'interpréter un grand nombre d'images en très peu de temps, offrant ainsi au patient, un traitement plus efficace et rapide.

Les réseaux de neurones convolutifs s'imposent aujourd'hui comme le type de réseau par excellence dans la classification d'images. De nouveaux types de réseaux neurones et de nouvelles méthodes apparaissent afin d'optimiser les performances de classification, comme par exemple le Transfer Learning.

Dans le cadre de ce projet, nous avons développé un modèle de Deep Learning afin de prédire si une radiographie de thorax provient d'un patient non malade ou d'un patient atteint de pneumonie.

Matériels et méthodes

I. Données

Nous avons à notre disposition un jeu de données [2] organisé en 3 dossiers : "train" pour l'apprentissage du modèle, "test" pour tester le modèle et "val" pour valider le modèle. Dans chacun de ses dossiers, nous avons deux sous-dossiers "NORMAL" et "PNEUMONIA" qui contiennent respectivement des radiographies de thorax sain et des radiographies de thorax atteint de pneumonie. Au total, nous avons 5 863 images répartis dans ces dossiers et sous-dossiers, enregistrées au format .JPEG.

Les radiographies de thorax ont été réalisées sur des patients pédiatriques âgés entre 1 et 5 ans de l'hôpital Guangzhou Women and Children's Medical Center situé à Guangzhou en Chine.

Les images ont été soumises à un contrôle de qualité. Toutes les radiographies de mauvaise qualité ou illisible ont été éliminées lors de ce contrôle. Le diagnostic des radiographies a été réalisé par deux médecins de l'hôpital.

II. Librairies

Nous avons utilisé différentes librairies afin de réaliser notre programme. Nous avons utilisé la librairie cv2 pour récupérer nos données et la librairie Keras afin de réaliser notre réseau de neurones.

III. Lecture et pré-traitement des données

Nous avons utilisé des méthodes de la librairie cv2 sur nos données, afin de lire les images de radiographie et les redimensionner afin qu'elles aient toutes la même dimension. Nous stockons nos images dans un array. De plus, nous avons normalisé notre array d'images en le divisant par 255. Nous avons effectué un random.shuffle() sur nos données afin de mélanger nos données de radiographies de thorax sains aux radiographies de pneumonies.

IV. Reconstruction du jeu de données de validation

Avant d'entraîner notre modèle, nous avons décidé de reconstruire notre jeu de données de validation. En effet, ce jeu de données, étant très petit, ne nous permet pas de valider correctement notre modèle. Nous avons donc concaténé le jeu d'apprentissage et le jeu de validation. Puis, nous avons reconstruit le jeu d'apprentissage en prenant 90% des données et les 10% de données restantes forment le jeu de validation.

V. Data augmentation

La data augmentation est une technique permettant d'augmenter artificiellement la quantité des données de notre jeu de données d'apprentissage et de validation. Cela est possible en modifiant l'aspect de nos images, sans modifier la sémantique, avec des opérations comme des rotations, des décalages ou des zooms. En entraînant notre modèle sur un plus grand jeu de données d'apprentissage, cela permet d'améliorer les performances et les capacités du modèle à généraliser l'apprentissage sur de nouvelles images.

Grâce à la librairie keras, cela est possible avec la classe ImageDataGenerator(). Dans notre cas, nous avons effectué des opérations de rotation, de zoom, de cisaillement, de retournement horizontal, de décalage en largeur et en longueur et de remplissage des zones décalées.

VI. Implémentation du réseau neuronal convolutif (CNN)

Nous avons implémenté un réseau neuronal convolutif composé de 13 couches afin de classifier nos radiographies en fonction de si elles proviennent de patients sains ou de patients atteints de pneumonie.

L'entraînement de notre modèle se fait sur 15 itérations, également appelées epochs. Lors d'une epoch, un passage complet dans l'ensemble des données d'apprentissage est effectué.

a) Couche de convolution Conv2D

Nous avons utilisé 4 couches de convolution Con2D. La première couche de convolution apprendra les petits motifs. Les couches de convolution suivantes apprendront des motifs de plus en plus grands. Cela permet d'apprendre des informations de plus en plus complexes. Nous avons choisi un nombre de filtres, respectivement 32, 64, 128 et 64 filtres pour nos 4 couches de convolution. Pour chaque couche de convolution, un motif de taille 3x3 pixels, également appelé kernel, est utilisé. Nous avons choisi la fonction ReLu comme fonction d'activation afin de laisser passer les valeurs positives aux couches intermédiaires suivantes du réseau de neurones. Chaque couche de convolution passe sur chaque pixel des images pour extraire des motifs de tailles 3x3 pixels et cette opération est réalisée autant de fois qu'il y a de filtres dans la couche. Sur toutes nos couches de convolution, nous n'avons pas utilisé de padding. La couche de convolution Conv2D prend en entrée une image en 3 dimensions (hauteur, largeur, couleur), soit un 3D-tensor et retourne une feature-map, qui est également un 3D-tensor.

b) Couche de pooling MaxPooling2D

Après avoir utilisé une couche Conv2D, nous utilisons une couche de pooling avec MaxPooling2D afin de réduire le résultat obtenu en sortie de la couche de convolution. La couche de pooling est également une couche de convolution, mais elle permet d'extraire la valeur la plus importante de chaque motif des features-map. La couche de pooling s'adapte à la couche précédente : nous n'avons donc pas besoin de lui indiquer un nombre de filtres. Elle extrait la valeur la plus importante de chaque motif de dimension 2x2 pixels.

c) Couche Flatten

Nous avons également mis une couche Flatten. Nous avons donné en entrée à notre modèle une image en 3 dimensions (hauteur, largeur, couleur) avec un 3D-tensor. Nous souhaitons retourner en sortie un label détecté, autrement dit un 1D-tenseur. La couche Flatten permet d'aplatir le tenseur, afin de réduire sa dimension de 3 à 1.

d) Couche Dropout

La couche Dropout a pour objectif de réduire le sur-apprentissage lors de l'entraînement du modèle. Des neurones vont être désactivés de façon aléatoire : nous avons choisi une probabilité p = 0.5 afin de désactiver les neurones. La désactivation est effectuée à chaque epoch, donc la configuration sera à chaque fois différente.

e) Couche Dense

La couche Dense permet d'obtenir le label détecté par notre modèle. Nous avons choisi la fonction d'activation softmax afin de donner une probabilité en sortie de chaque neurone. La somme des probabilités de tous les neurones de la couche doit être égale à 1. Le neurone avec la plus grande probabilité permet de décider le label.

f) Compilation du modèle

Nous avons compilé notre modèle avec la fonction de perte “sparse_categorical_crossentropy”. Cette fonction permet de retourner le label le plus probable de l’image. Nous avons choisi l’optimiseur “adam” pour la compilation de notre modèle. L’optimiseur permet de mettre à jour les paramètres des poids afin de minimiser la fonction de perte. La fonction de perte indique à l’optimiseur s’il se déplace dans la bonne direction pour atteindre le minimum global.

VII. Implémentation de la méthode Transfer Learning

Nous avons essayé d’implémenter un transfer learning sur notre réseau de neurones. Cette technique permet de faciliter l’apprentissage de notre modèle, en effectuant un apprentissage précédent. Grâce à cela, le modèle final est capable de reconnaître de façon plus précise de nouvelles images. Nous n’avons pas réussi à pré-entraîner notre modèle, mais nous avons essayé les deux techniques suivantes de Transfer Learning :

a) ResNet50

Les réseaux ResNet sont formés sur l’ensemble de données ImageNet. Il y a 50 couches de neurones. On modifie les dernières couches de ce modèle qui a déjà appris afin d’ajouter notre modèle d’apprentissage. Ce pré-apprentissage permet d’obtenir de meilleures performances et une meilleure prédiction des labels des images.

b) VGG16

VGG16 est un réseau neuronal convolutif atteignant une précision de 92.7% de précision dans la base de données d’ImageNet.

Le réseau prend en entrée une image RGB de taille 224x224 pixels. L'image est passée à travers un empilement de couches convolutives où les filtres ont été utilisés avec un très petit champ récepteur : 3×3. Trois couches entièrement connectées (FC) suivent une pile de couches convolutives du modèle excepté la dernière couche. On gèle les poids du réseau VGG16 pour qu’il n’y ai plus d’apprentissage et de perte d’information sur les données apprises par le réseau. On rajoute une couche de type Flatten afin d'aplatir nos données et une couche de type Dense afin d’effectuer la classification finale et déterminer si notre poumon est sain ou malade.

Résultats

I. Présentation des données

Nous avons réalisé des barplots sur nos jeux de données d’apprentissage, de test et de validation.

Figure 1 : Barplot des données d'apprentissage

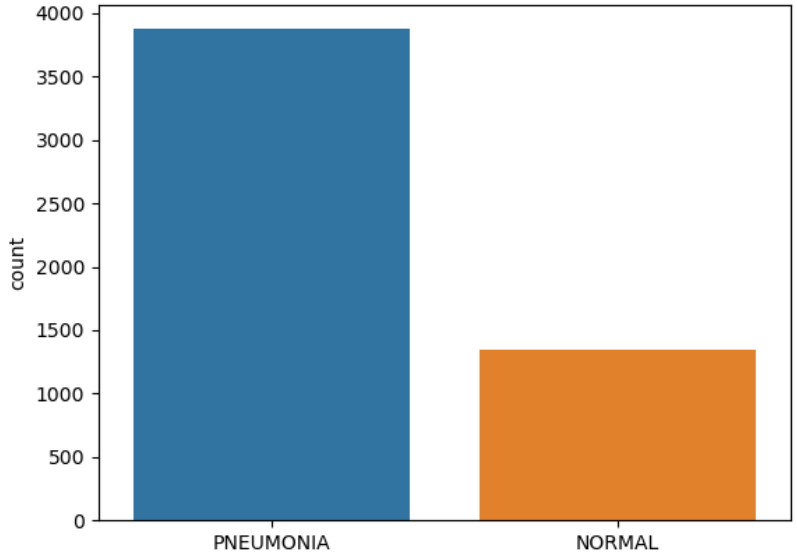


Figure 2 : Barplot des données de test

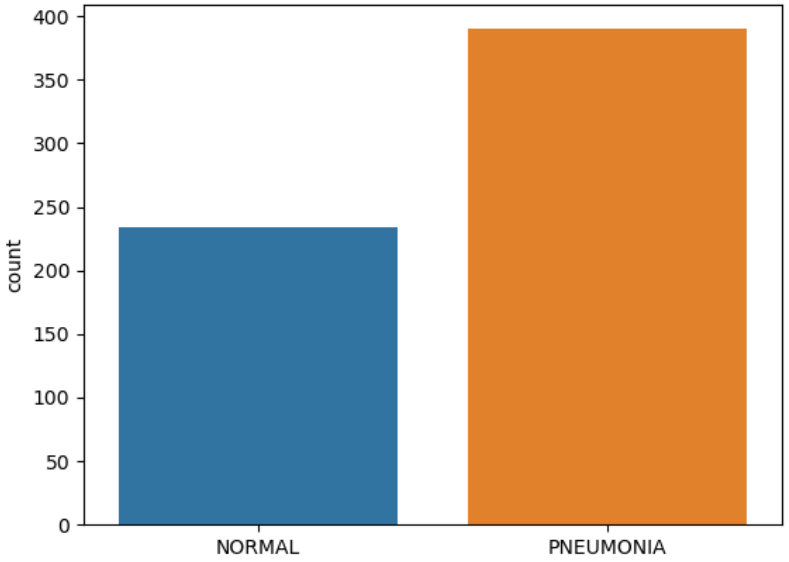
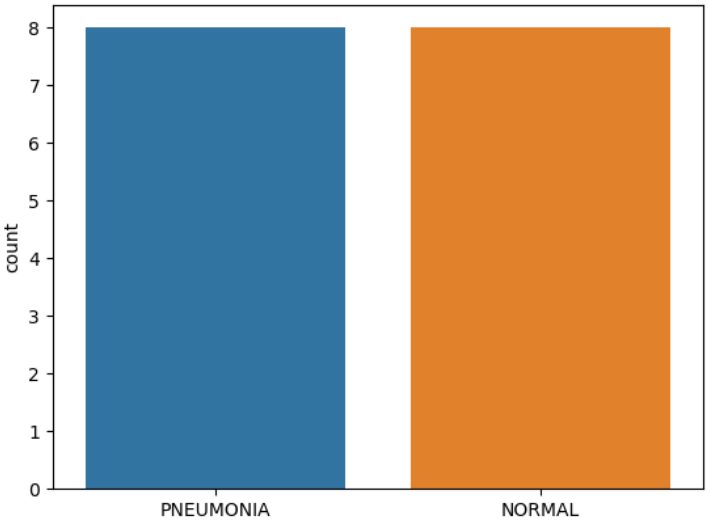


Figure 3 : Barplot des données de validation



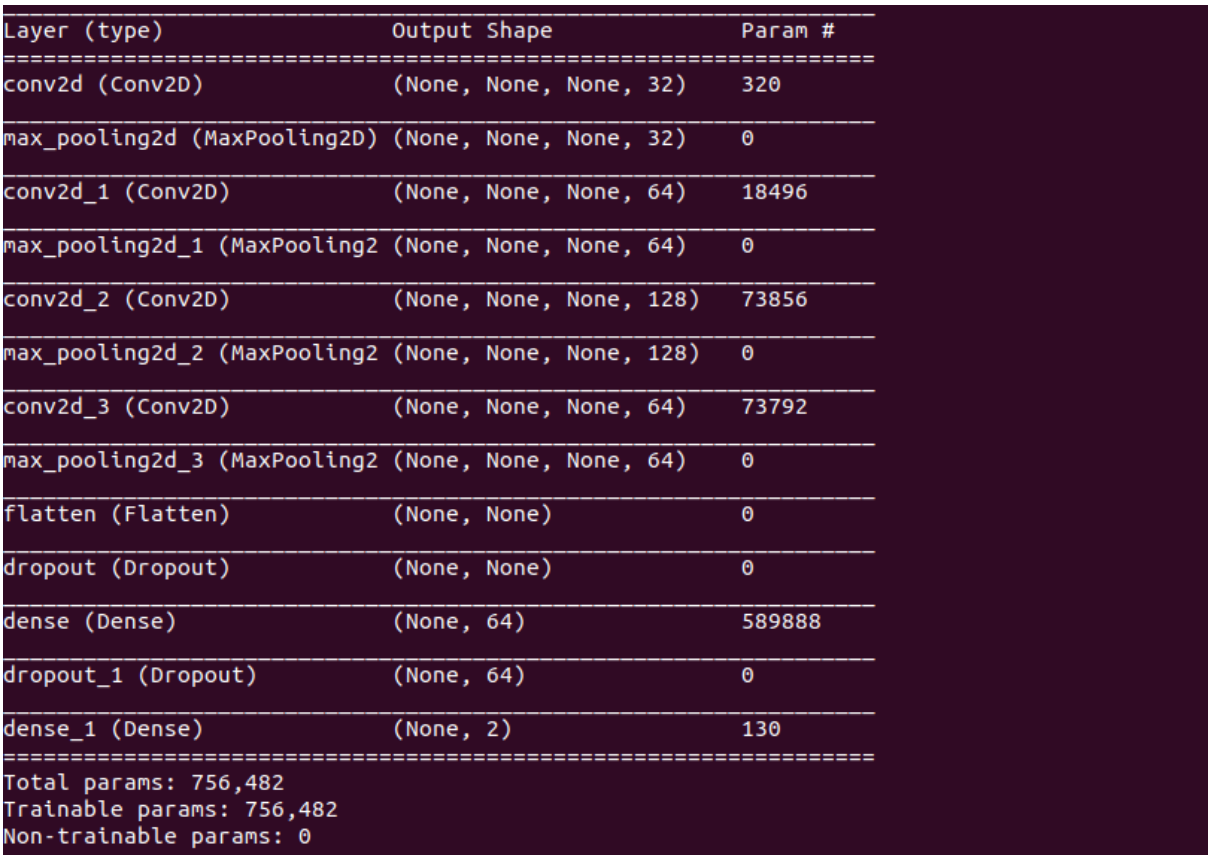
Sur la figure 1, nous pouvons observer que la quantité de données de radiographies de poumons atteint de pneumonie est supérieure à celle de poumons sains. Cela reste globalement du même ordre de grandeur, donc nous avons décidé de ne pas ajouter de poids à nos données pour les équilibrer.

Sur la figure 2, nous pouvons constater que le jeu de données de validation est très petit. Nous avons à notre disposition un jeu de 16 images seulement pour valider notre modèle. Au vu de cette faible taille de jeu de données, nous avons décidé de reconstruire le jeu de validation. Pour cela nous l'avons concaténer au jeu d'apprentissage, puis nous avons re-construit nos jeux en sélectionnant 90% des données pour former le jeu d'apprentissage et les 10% de données restantes pour le jeu de validation. Puis, nous avons effectué une data augmentation sur le jeu d'apprentissage et de validation afin d'augmenter nos données de façon artificielle.

II. Modèle d'apprentissage

Après avoir entraîné notre modèle sur 15 epochs, nous obtenons le modèle de la figure 4. Nous retrouvons nos 13 couches de neurones. Un nombre de paramètres a été calculé. Ce nombre de paramètres correspond au nombre de poids pour chaque couche.

Figure 4 : Capture d'écran du summary() du modèle sur le terminal



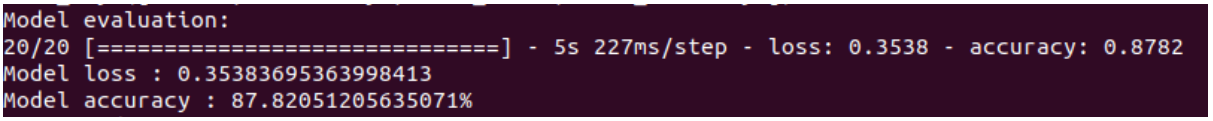
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, None, None, 32)    320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, None, None, 32)    0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, None, None, 64)    18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, None, None, 64)    0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, None, None, 128)   73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, None, None, 128)   0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, None, None, 64)    73792
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, None, None, 64)    0
_________________________________________________________________
flatten (Flatten)            (None, None)              0
_________________________________________________________________
dropout (Dropout)            (None, None)              0
_________________________________________________________________
dense (Dense)                (None, 64)                589888
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 756,482
Trainable params: 756,482
Non-trainable params: 0
```

Au total pour ce modèle, nous avons calculé 756 482 poids sur nos 13 couches. Ce summary permet de confirmer la sortie de chaque couche et de vérifier l'ordre de nos différentes couches.

III. Résultats sur le jeu de test

Après l'apprentissage de notre modèle, nous avons évalué ses performances sur le jeu de données test grâce à la méthode evaluate() de Keras. Nous obtenons une précision de 87.82%. En d'autres termes, notre modèle arrive à détecter le bon label d'une image de thorax dans 87.82% des cas. Nos performances sur le jeu de données test sont bonnes.

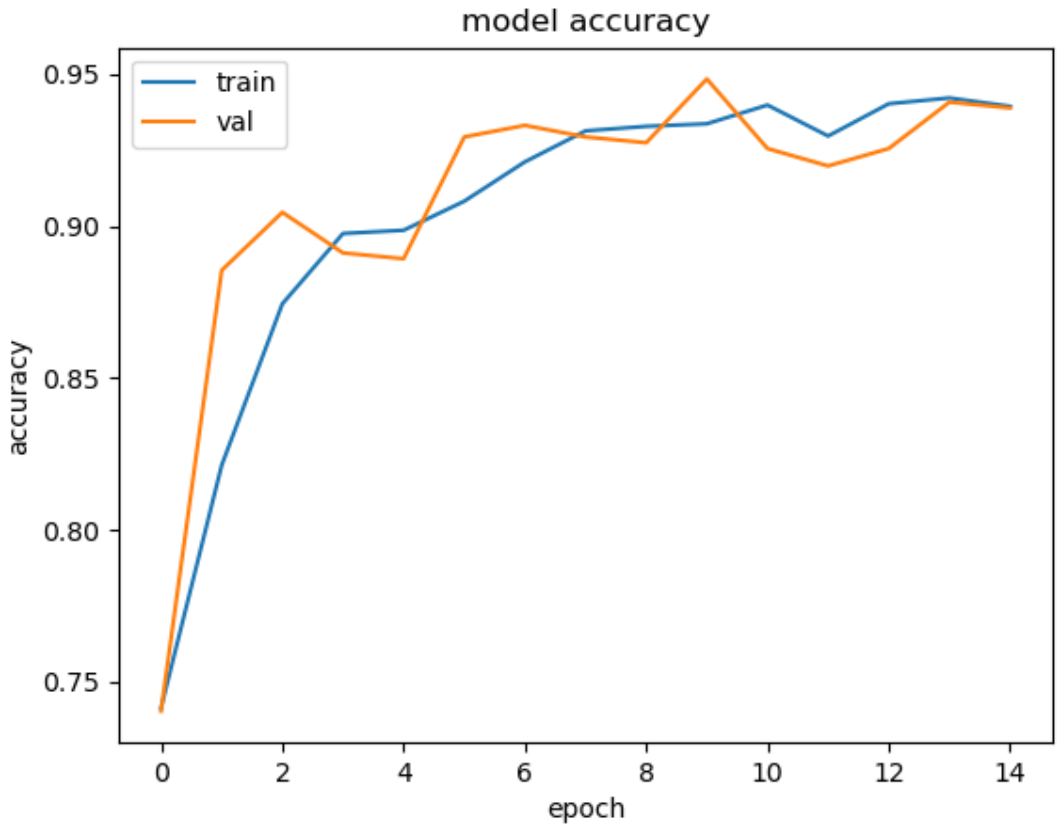
Figure 5 : Capture d'écran des résultats sur les données de test sur le terminal



```
Model evaluation:
20/20 [==============================] - 5s 227ms/step - loss: 0.3538 - accuracy: 0.8782
Model loss : 0.35383695363998413
Model accuracy : 87.82051205635071%
```

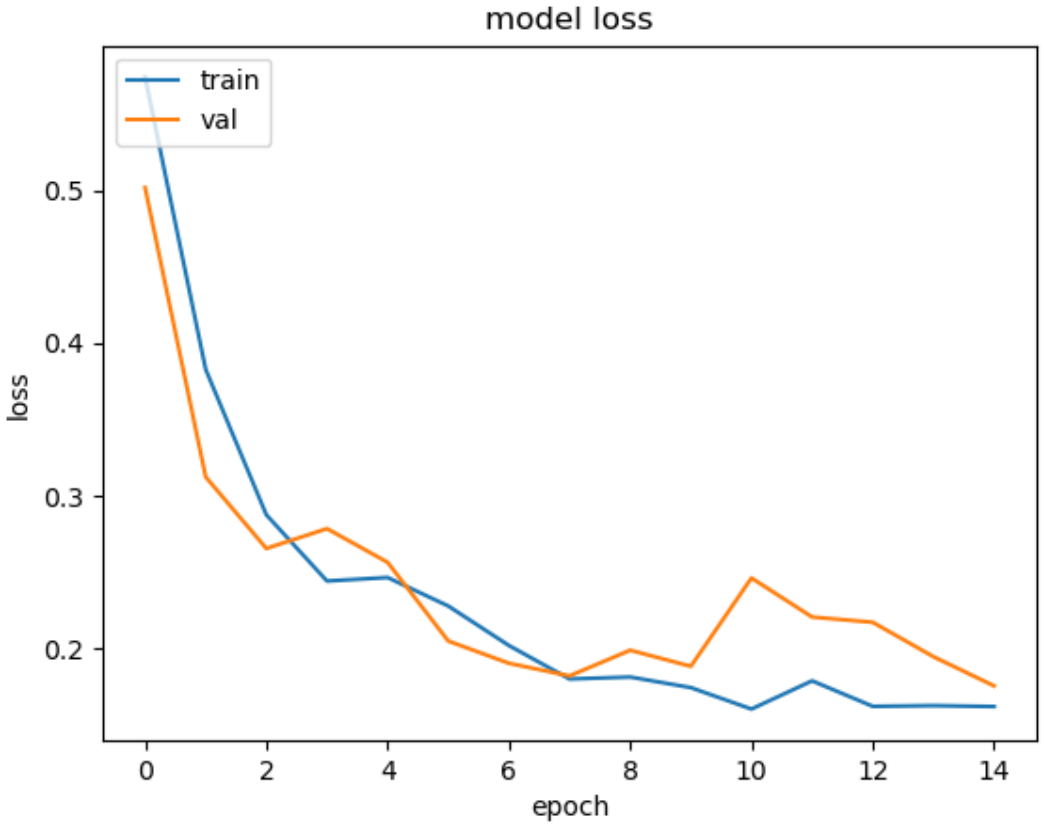
Nous avons obtenu la courbe de précision de la figure 6. Au cours des itérations, on observe que la précision sur le jeu de données d'apprentissage augmente. Sur le jeu de validation, la précision augmente également et les deux courbes sont relativement proches. Notre modèle apprend correctement sur le jeu de données d'apprentissage. Et, il est capable de reproduire la même précision sur le jeu de validation.

Figure 6 : Courbe de précision (accuracy) de notre modèle



Au cours des epochs, nous observons sur la figure 7 que la perte diminue. Cela nous confirme que notre modèle apprend correctement. Sur le jeu de validation, la perte diminue également, malgré quelques variations. Ces variations auraient pu être diminuées avec un jeu de validation plus grand.

Figure 7 : Courbe de perte de notre modèle



Discussion

Notre modèle apprend correctement sur notre jeu d'apprentissage. Nous avons obtenu de bonnes performances sur le jeu de test. Grâce à la reconstruction du jeu de validation, nous avons obtenu de bons résultats.

Conclusion

Au cours de ce projet, nous avons réussi à implémenter un réseau de neurones convolutif capable de prédire si la radiographie de thorax provient d'un patient sain ou d'un patient atteint de pneumonie avec une précision de 87.82%. Notre modèle a appris correctement sur le jeu d'apprentissage et nous obtenons de bons résultats sur le jeu de validation.

Bibliographie

[1] Kermany, D. S., Goldbaum, M., Cai, W., Valentim, C. C. S., Liang, H., Baxter, S. L., McKeown, A., Yang, G., Wu, X., Yan, F., Dong, J., Prasadha, M. K., Pei, J., Ting, M. Y. L., Zhu, J., Li, C., Hewett, S., Dong, J., Ziyar, I., … Zhang, K. (2018). Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning. In Cell (Vol. 172, Issue 5, pp. 1122-1131.e9). Elsevier BV. https://doi.org/10.1016/j.cell.2018.02.010

[2] Kermany, D. (2018). Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification [Data set]. Mendeley. https://doi.org/10.17632/RSCBJBR9SJ.2

Annexes

Historique des versions

Notre rendu final est disponible sur *mon git*. Les précédentes versions de notre code sont disponibles sur les liens suivants : https://github.com/BerkaneSam/KagglePneumo.git et https://colab.research.google.com/drive/1t5Boo7-wifzVH6qR-4Ae-SS4GRF5rIlf?usp=sharing