T-DEV-810 : ZOIDBERG

Introduction :

Le projet ZoidBerg consiste en la réalisation d'un modèle d'intelligence artificielle servant à reconnaître des radiographies de patient atteint de Pneumonie. Nous avons essayé et comparé trois méthodes pour arriver à ce but : le réseau neuronal dense, le réseau neuronal convolutif et l'algorithme de random forest.

Contexte du groupe :

Nous étions un groupe de 3 membres, mais l'un d'entre nous a eu des problèmes personnels l'empêchant de travailler avec le reste du groupe. Le projet a donc été réalisé par deux personnes, impliquant un déficit de temps sur le projet. La troisième personne à quant à elle mené ses propres recherches de son côté afin de tout de même être évaluée.

Méthode Dense :

La méthode Dense est un modèle d'intelligence artificielle qui utilise plusieurs couches de neurones pour trouver un résultat. Pour cet algorithme, nous avons porté notre attention sur deux paramètres en particulier :

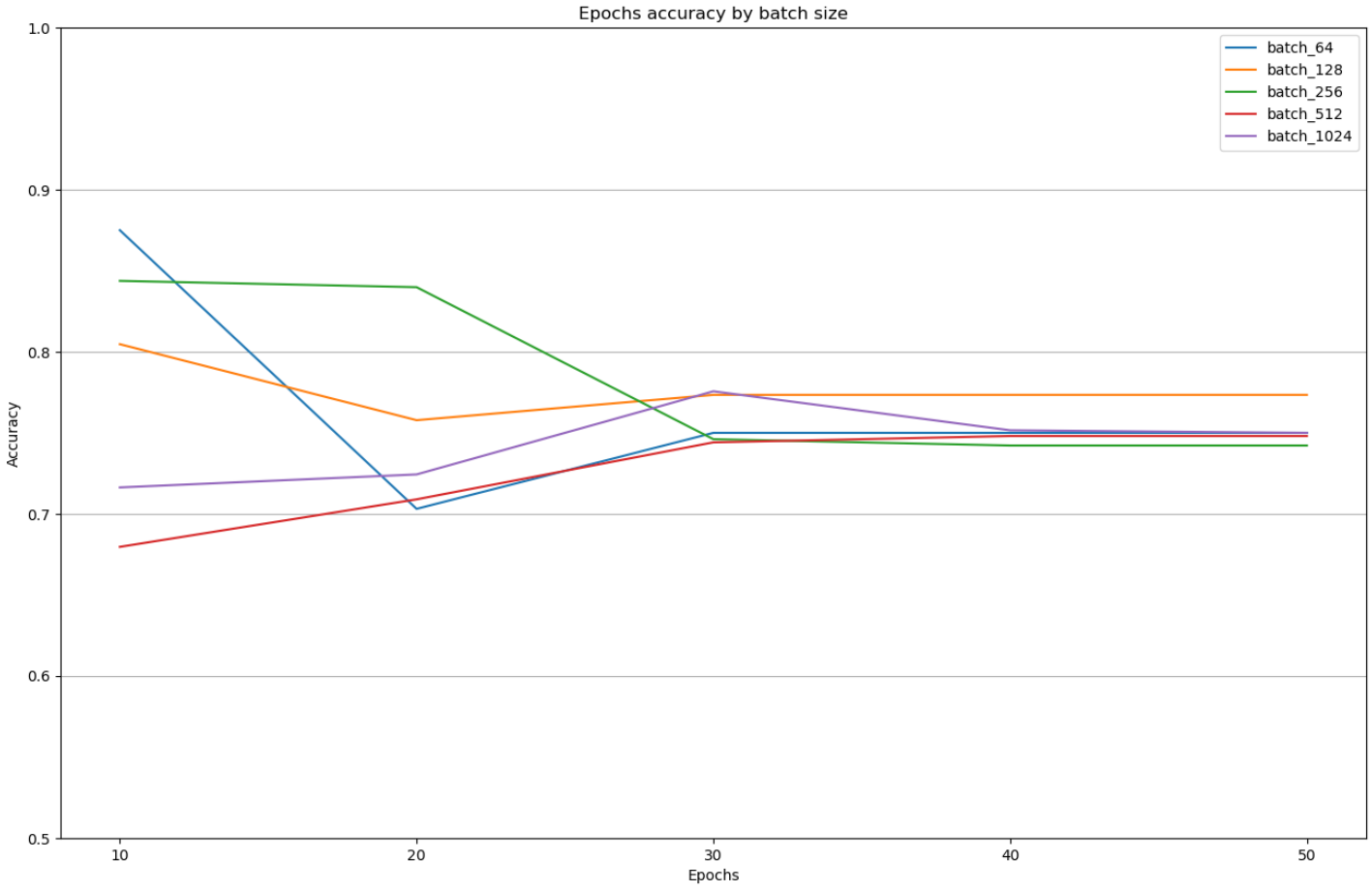
- batch_size

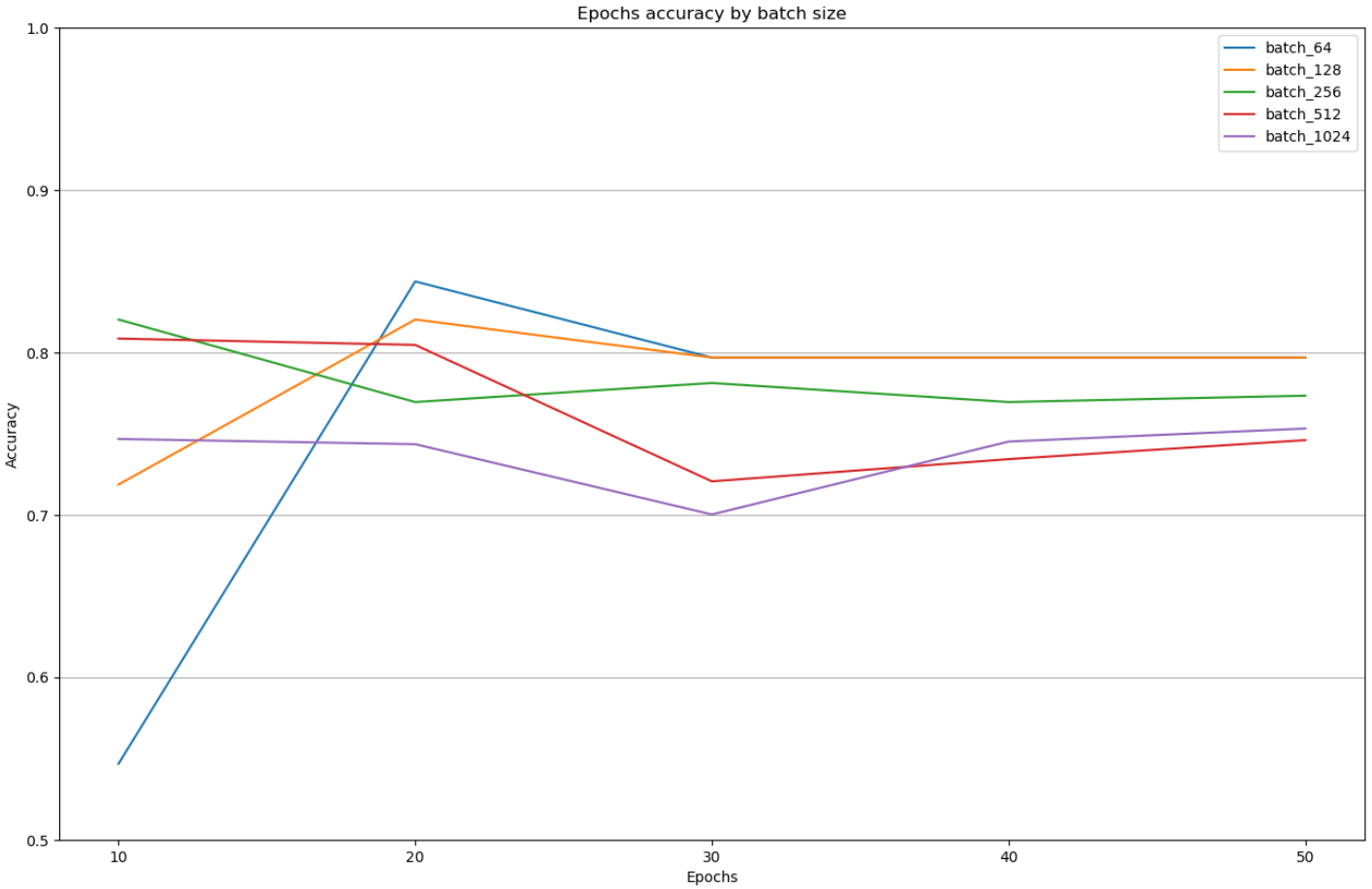
- epochs

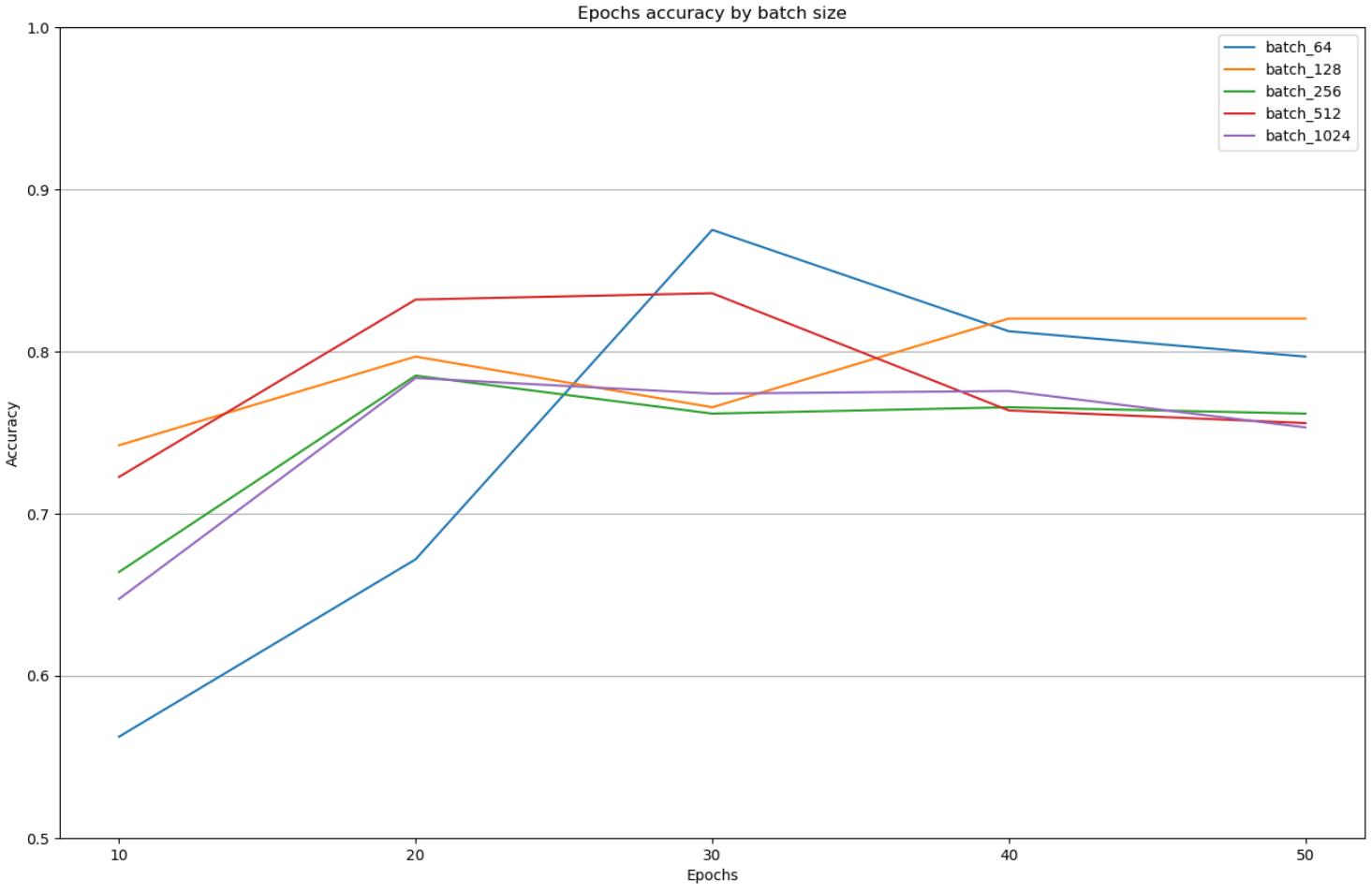
Le batch_size représente la taille des paquets de données de test à passer dans le réseau de neurones à chaque epoch. L'epochs est donc le nombre de paquets passés dans le réseau de neurones pour un entraînement. Plus ces paramètres sont grands, plus l'entraînement est long.

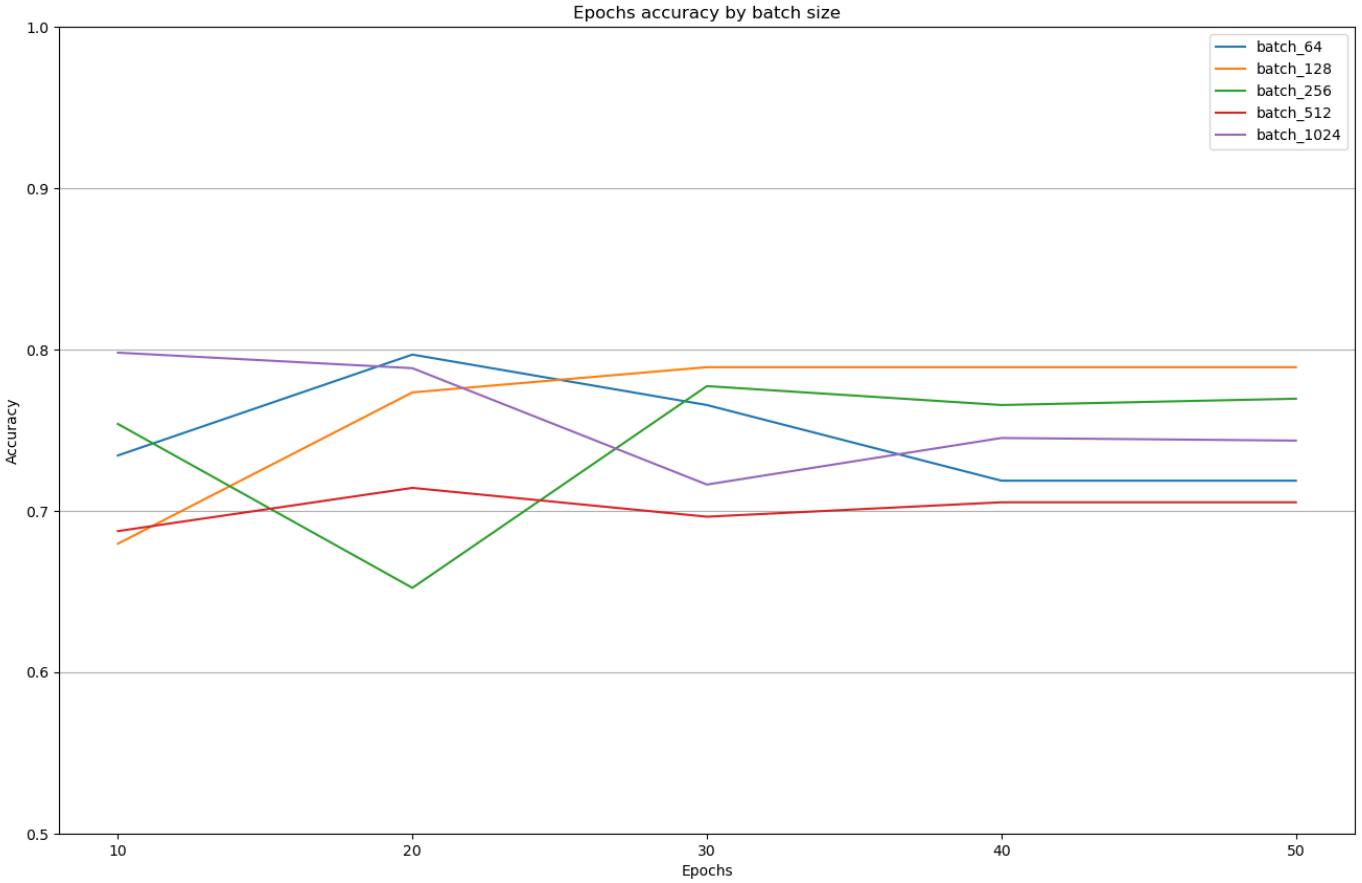
Pour ces tests, nous avons utilisé 3 couches dense : deux de 128 neurones et une de 2 neurones (car seulement deux résultats possible). Nous n'avons pas essayé d'augmenter le nombre de couches ou le nombre de neurones par manque de temps.

Le graphique suivant montre l'évolution, en fonction de l'epochs, de la précision des entraînements avec différents batch_size. Chacun des graphiques a été fait avec les mêmes paramètres.




Epochs accuracy by batch size
batch_64
batch_128
batch_256
batch_512
batch_1024
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
10
20
30
40
50


Epochs accuracy by batch size
batch_64
batch_128
batch_256
batch_512
batch_1024
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
10
20
30
40
50



Lorsqu'une courbe stagne à la même valeur, cela signifie que le modèle fait de l'overfitting, c'est-à-dire qu'il apprend par cœur les valeurs d'entraînement et ne peut ainsi plus s'améliorer. Le but est donc d'avoir une précision maximale avant que le modèle n'overfit.

Sur les graphiques, on observe qu'il n'y a pas d'overfitting pour des epochs < 30, peu importe le batch_size. Le batch_size ne semble pas avoir d'incidence sur l'overfitting.

Quant à la précision, on observe que celle-ci est très instable pour le batch_size = 64, mais qu'à partir de 128, plus le batch_size augmente, moins le modèle est précis.

Nous utiliserons donc un batch_size de 128 ainsi qu'une epochs situé entre 20 et 30 pour notre modèle final.

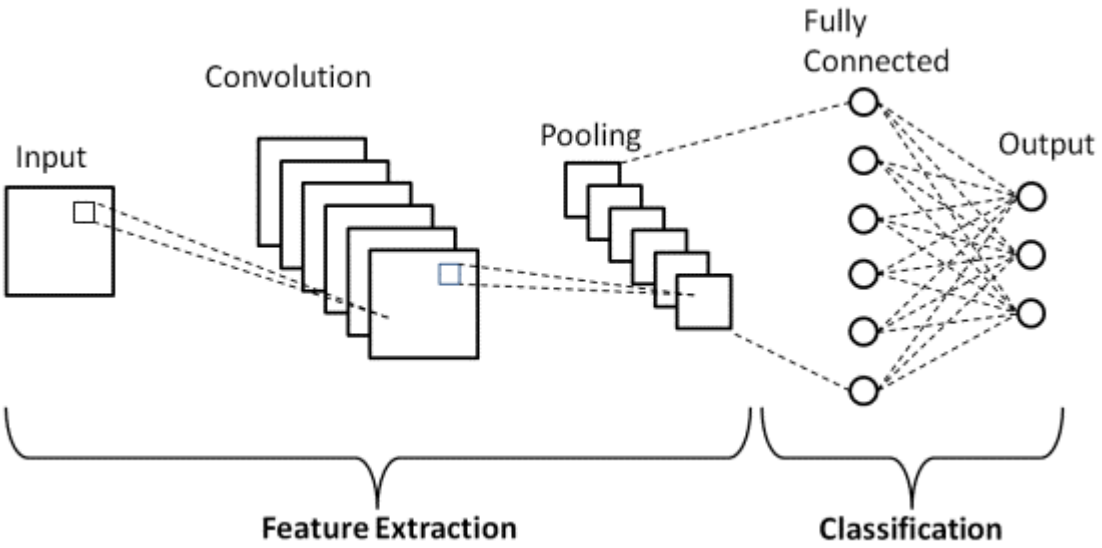
Nous obtenons donc une précision de 0.7812 avec un batch_size de 128 et une epochs de 20.

Méthode CNN :

Un Convolutional Neural Network est un type de réseau de neurones très utilisé dans le cadre de la reconnaissance d'image, de vidéo ou dans le traitement du langage naturel.

Nous avons décidé d'utiliser python avec la bibliothèque Keras pour le CNN. Après avoir choisi notre modèle, nous allons l'entrainer en faisant varier la batch size et le nombre d'epochs afin d'observer les variations d'accuracy que cela engendre.

Schéma d'un CNN



Le CNN se décompose en 2 parties, la première va permettre d'extraire des features de l'image (des informations) via des couches Conv2D et MaxPooling2D. Ces features vont ensuite êtres reçues par une couche Dense qui va permettre la classification des images.

Schéma de fonctionnement d'une couche Conv2D

Cette couche est la clé du CNN, elle va réaliser un filtrage par convolution en faisant "glisser" le kernel sur l'image et en calculant le produit de convolution à chaque fois.

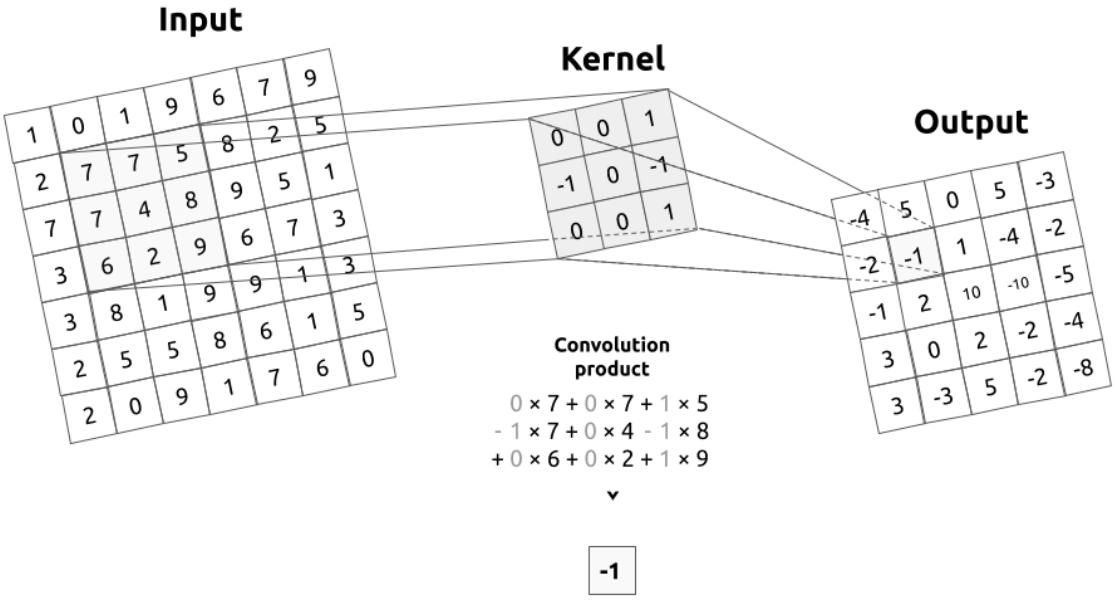
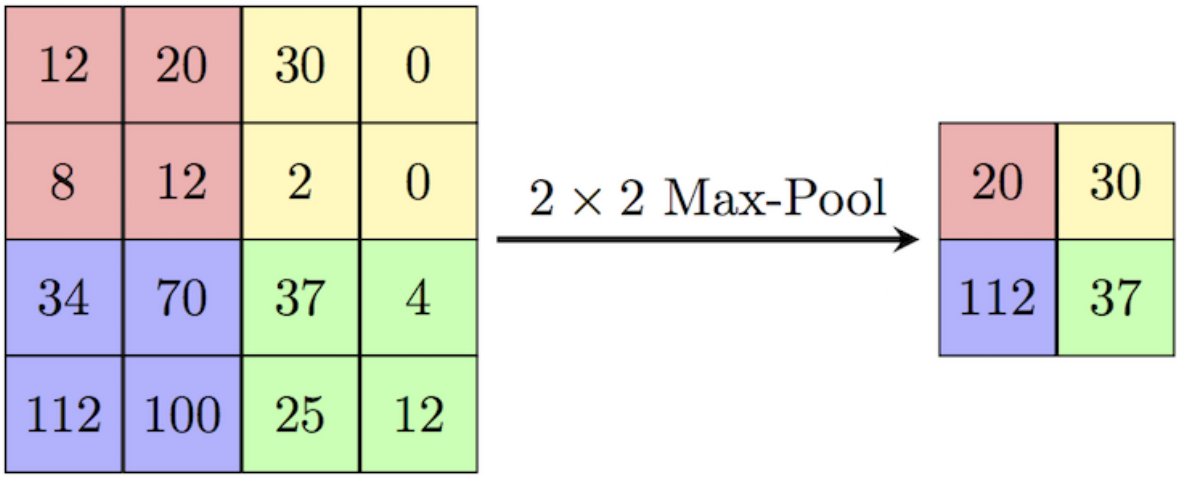


Schéma de fonctionnement d'une couche MaxPooling2D

Cette couche est souvent placée en sortie d'une couche de convolution, elle a pour but de réduire la taille de l'image tout en gardant les caractéristiques importantes. Pour cela, elle découpe l'image en plusieurs zones régulières et elle garde la valeur maximale de chaque zone.



Couche Dropout

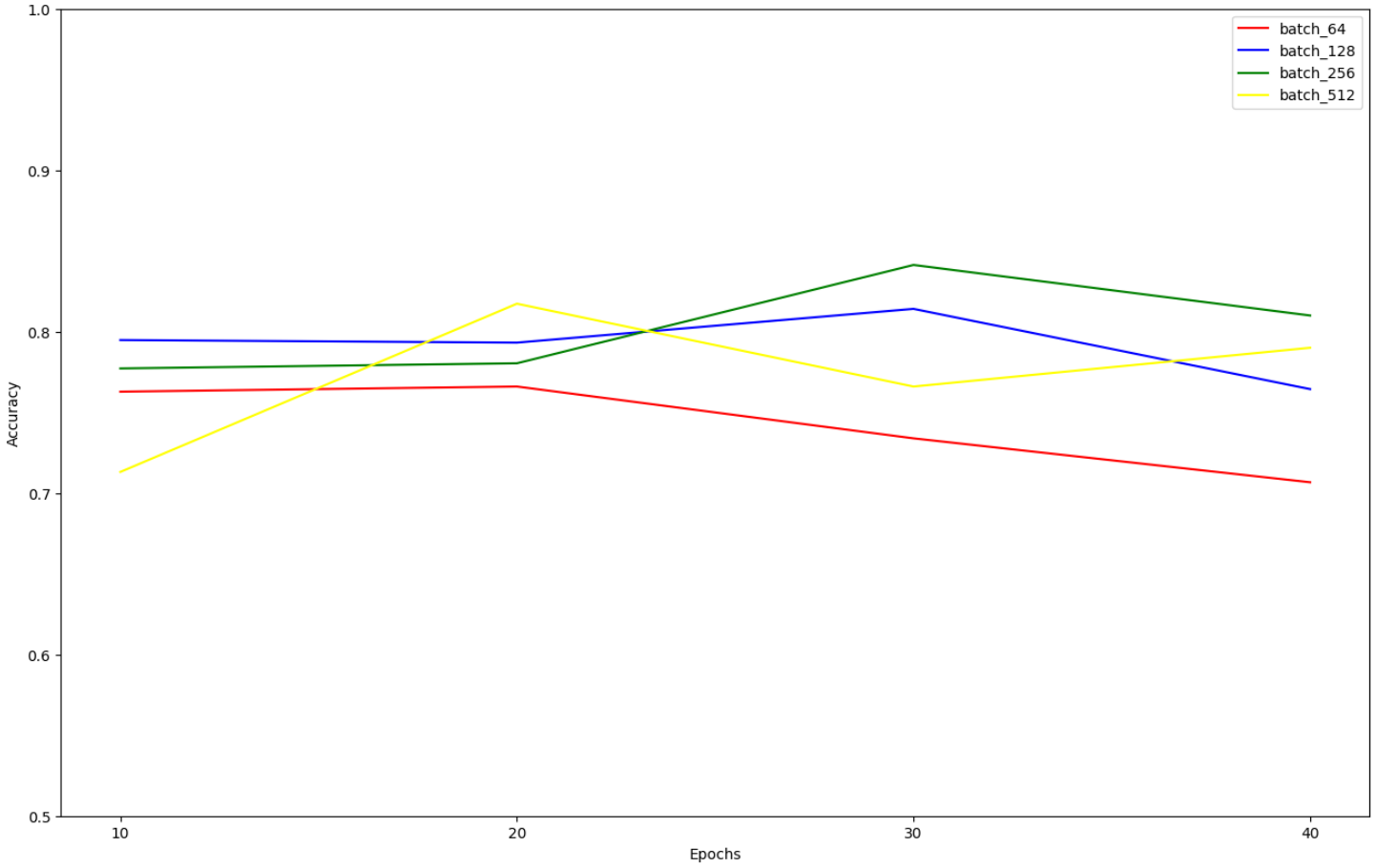
Nous avons utilisé des couches Dropout afin de diminuer l'overfitting et d'optimiser l'apprentissage. Les couches dropout permettent de mettre le poids de certains neurones à 0 aléatoirement et ainsi de les désactiver. Cela permet au réseau de neurones de réaliser un entraînement avec des neurones actifs différents à chaque epochs et ainsi éviter d'avoir un apprentissage trop similaire à chaque fois qui pourrait mener à l'overfitting.

Fonctionnement global CNN

L'objectif du CNN est donc de prendre en entrée une image et de réduire sa taille de plus en plus en ne gardant que les features importantes pour pouvoir ensuite procéder à la classification.

Résultats de notre modèle

Accuracy	10 epochs	20 epochs	30 epochs	40 epochs
batch_size 64	0.7628	0.7660	0.7339	0.7067
batch_size 128	0.7948	0.7932	0.8141	0.7644
batch_size 256	0.7772	0.7804	0.8413	0.8190
batch_size 512	0.7131	0.8173	0.7660	0.7981



Nous avons ici un graphique des différentes accuracy de notre modèle pour des batch size variant entre 64 et 512 et des nombres d'epochs variant entre 10 et 40.

La meilleur précision, 84% à été obtenue avec une batch size de 256 et 30 epochs

Cependant, on peut noter que nous n'avions pas d'overfitting avec 40 epochs et qu'il serait donc possible de pousser le nombre d'epoch plus loin avec potentiellement de meilleurs accuracy au prix du temps d'entraînement.

Algorithme RandomForest :

La méthode random forest est basé sur deux algorithmes de machine learning :

- Les arbres de décisions

- L'ensemble learning

Les arbres de décisions sont des modèles faciles à entraîner, mais peu précis et difficilement généralisables à de nouvelles données. C'est-à-dire qu'ils sont performants sur les données d'entraînements, mais beaucoup moins sur les données de test.

Pour pallier ce problème, l'algorithme du random forest fait appel à l'ensemble learning, qui consiste à utiliser plusieurs modèles différents et à combiner leurs résultats afin d'obtenir un résultat très fiable. Dans l'algorithme Random Forest, les différents modèles utilisés dans l'ensemble learning sont tous des arbres de décisions se focalisant sur des caractéristiques différentes, formant ainsi une forêt. C'est en combinant le résultat de chacun des arbres que l'on obtient une précision bien meilleure qu'avec un simple arbre de décision.

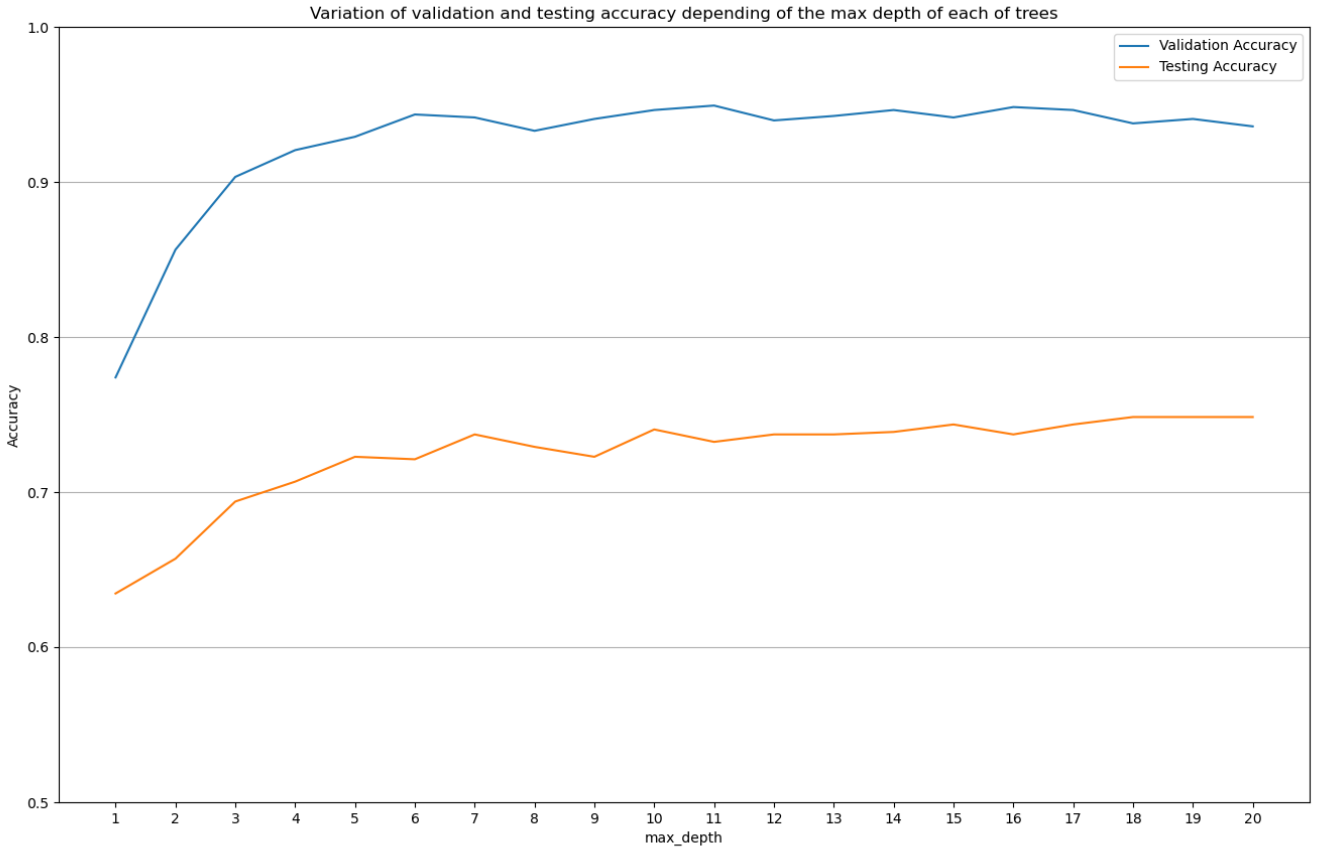
Pour utiliser cet algorithme, nous utiliserons la librairie python Sklearn.

Tous les tests présentés ont été réalisés avec le paramètre random_state = 100. Cela permet de toujours avoir les mêmes valeurs d'entraînement. Cela nous permet d'être sûrs que les changements de précision sont dus à la modification d'un paramètre et non à une variation des données d'entraînements.

Le paramètre n_estimators représente le nombre d'arbres dans la random forest, plus ce nombre est élevé plus les résultats seront précis, mais plus l'entraînement sera long. Pour les tests, nous utiliserons la valeur arbitraire n_estimators = 30.

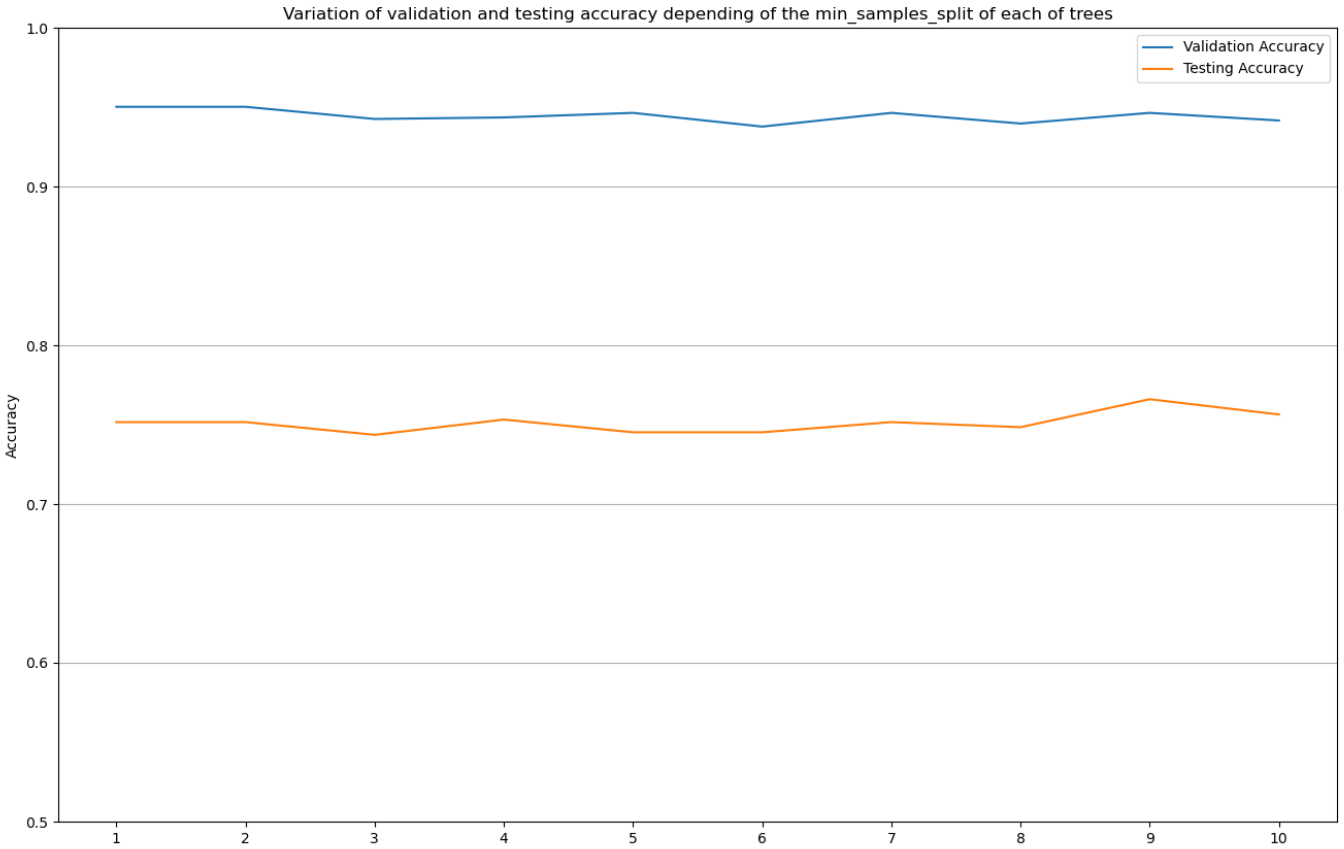
Avec RandomForestClassifier(n_estimators=30, random_state=100), nous obtenons une précision de 0.7548, ce qui sera notre précision de départ que nous chercherons à améliorer.

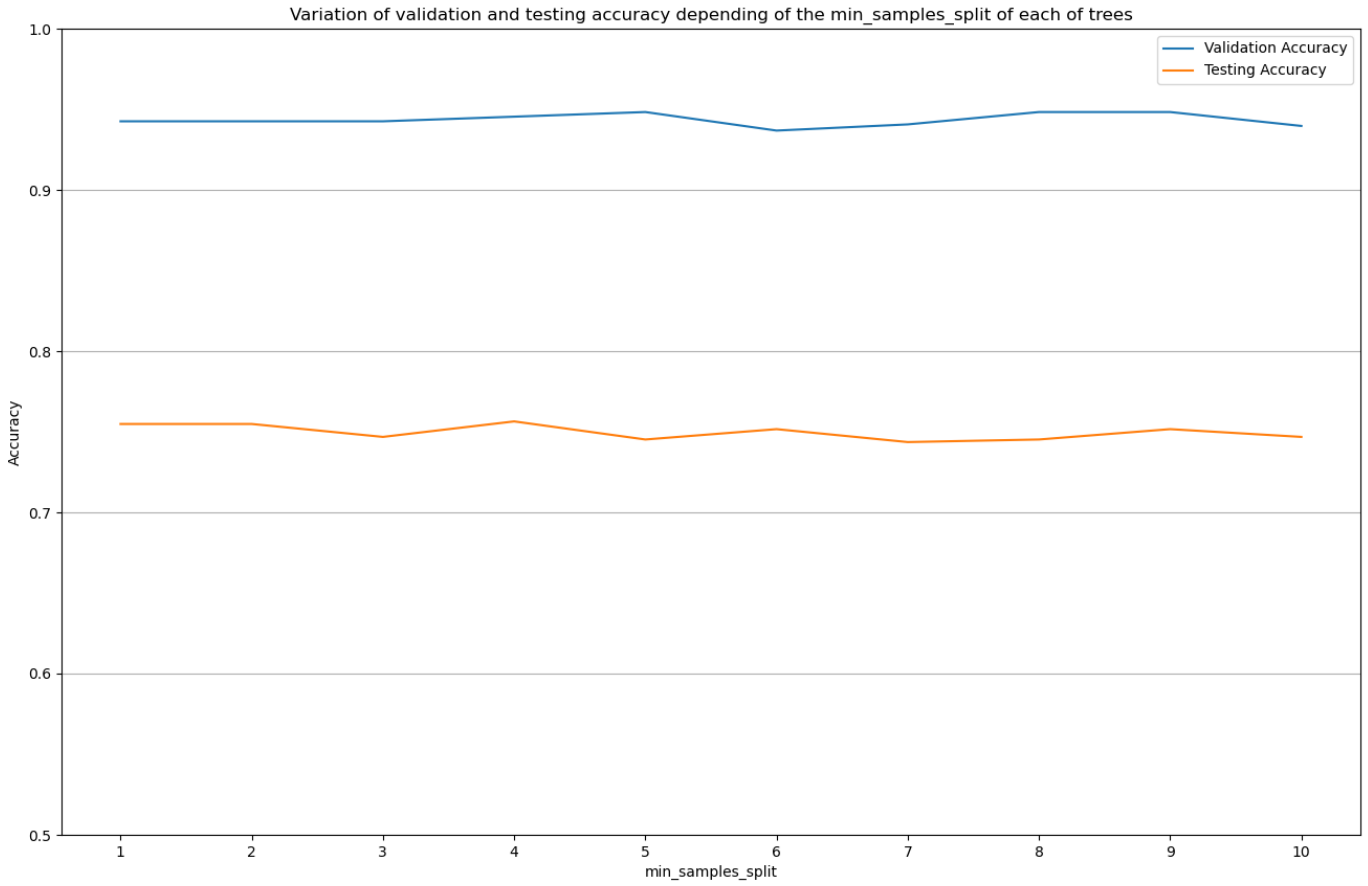
Déterminer le paramètre max_depth :



D’après le graphique ci-dessus la meilleure précision est obtenue pour max_depth = 18, 19 et 20 cependant la précision atteinte est 0.7483 ce qui est inférieur à la valeur de départ (0.7483 < 0.7548).

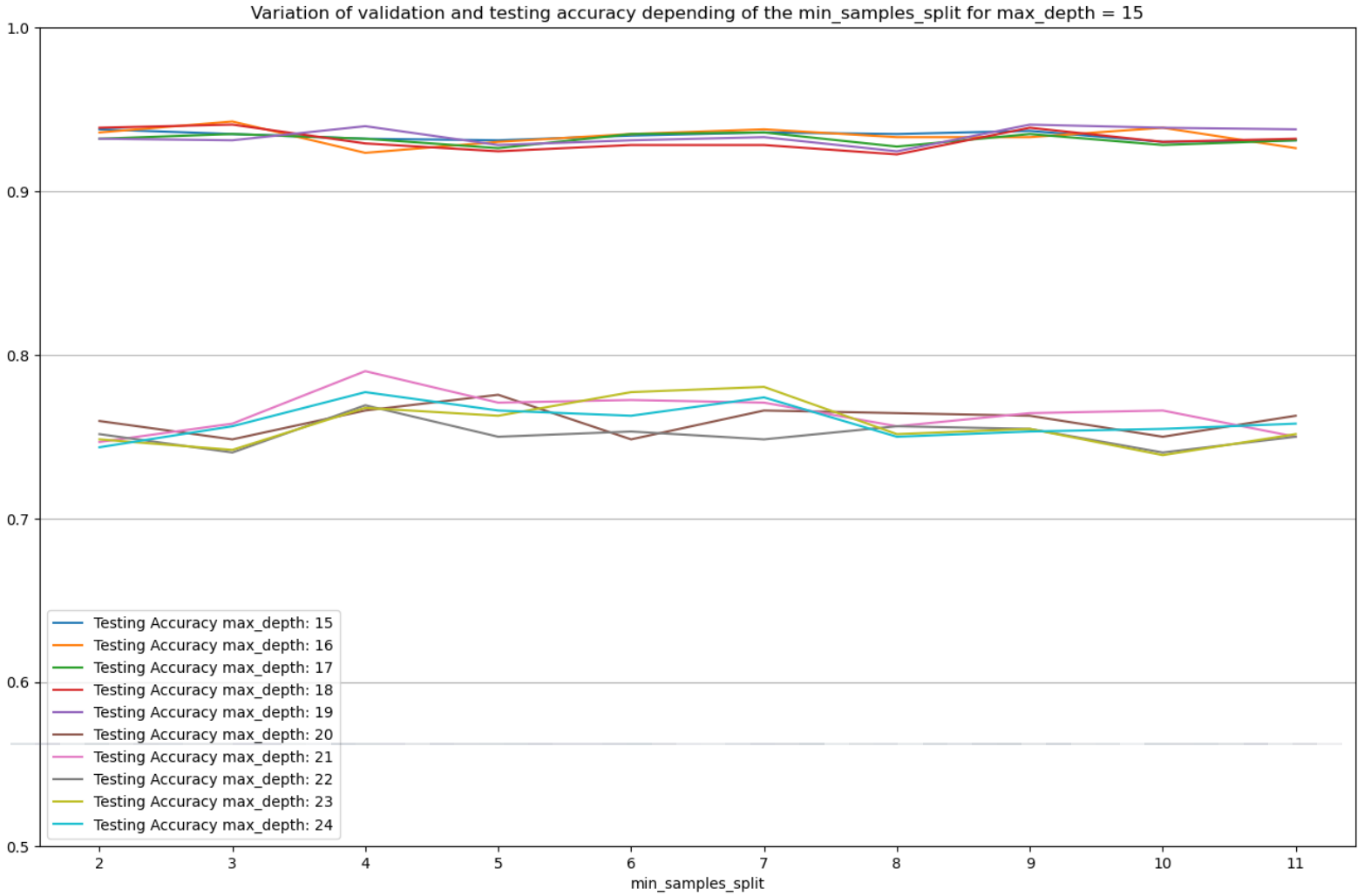
Déterminer min_samples_split :

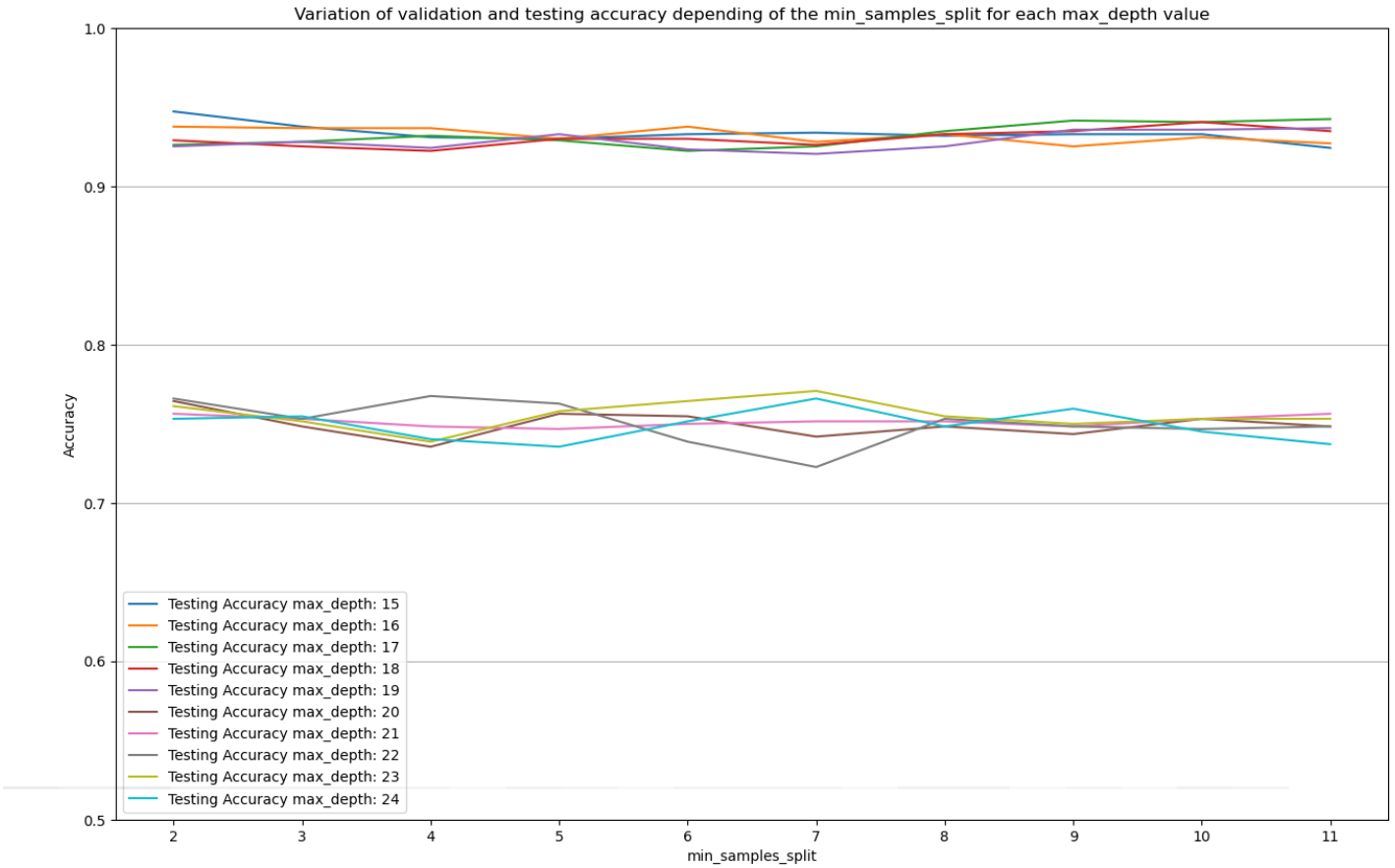




D’après les deux graphiques précédents, nous pouvons supposer qu’il n’est pas très utile de modifier la valeur de min_samples_split.

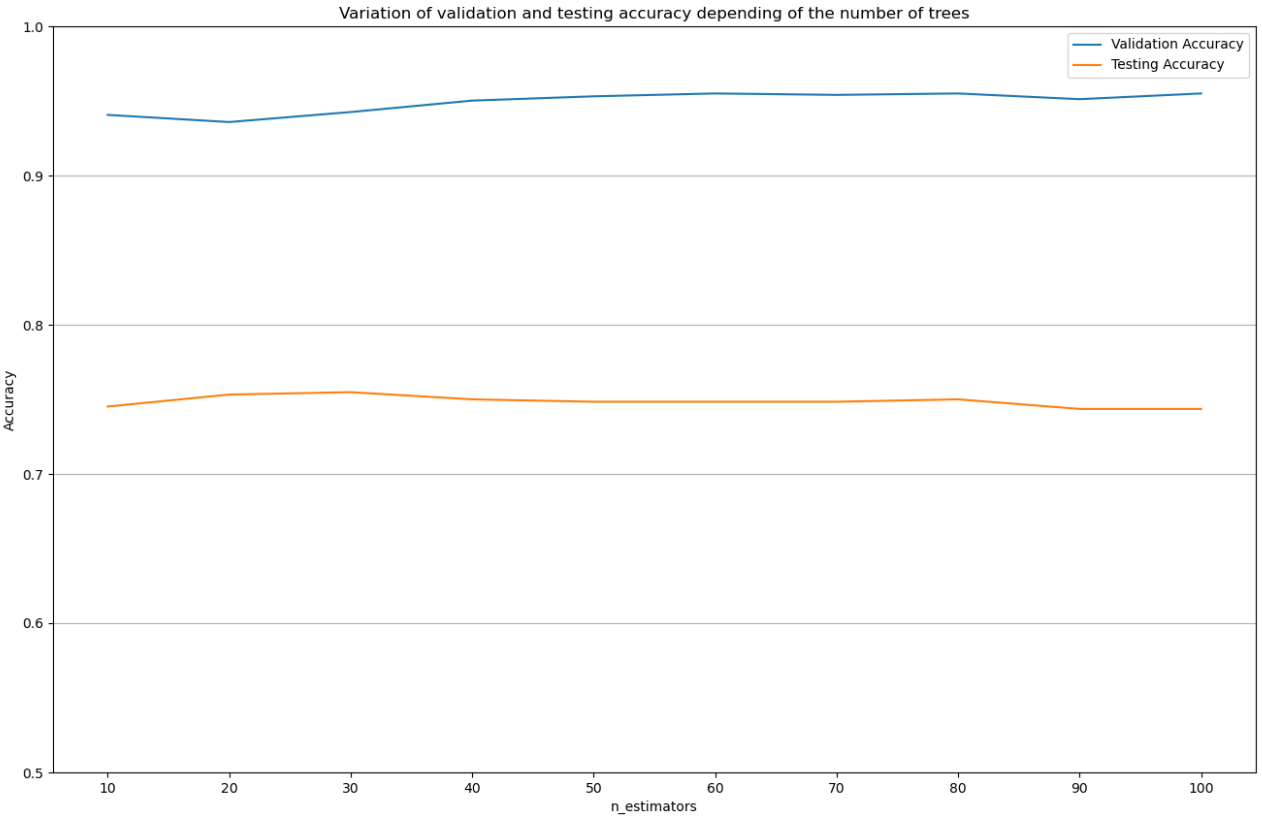
Nous avons cependant testé d’augmenter en parallèle min_samples_split et max_depth ce qui donne cette suite de graphiques, chacun montrant l’évolution de la précision en fonction de min_samples_split pour une valeur max_depth donnée :

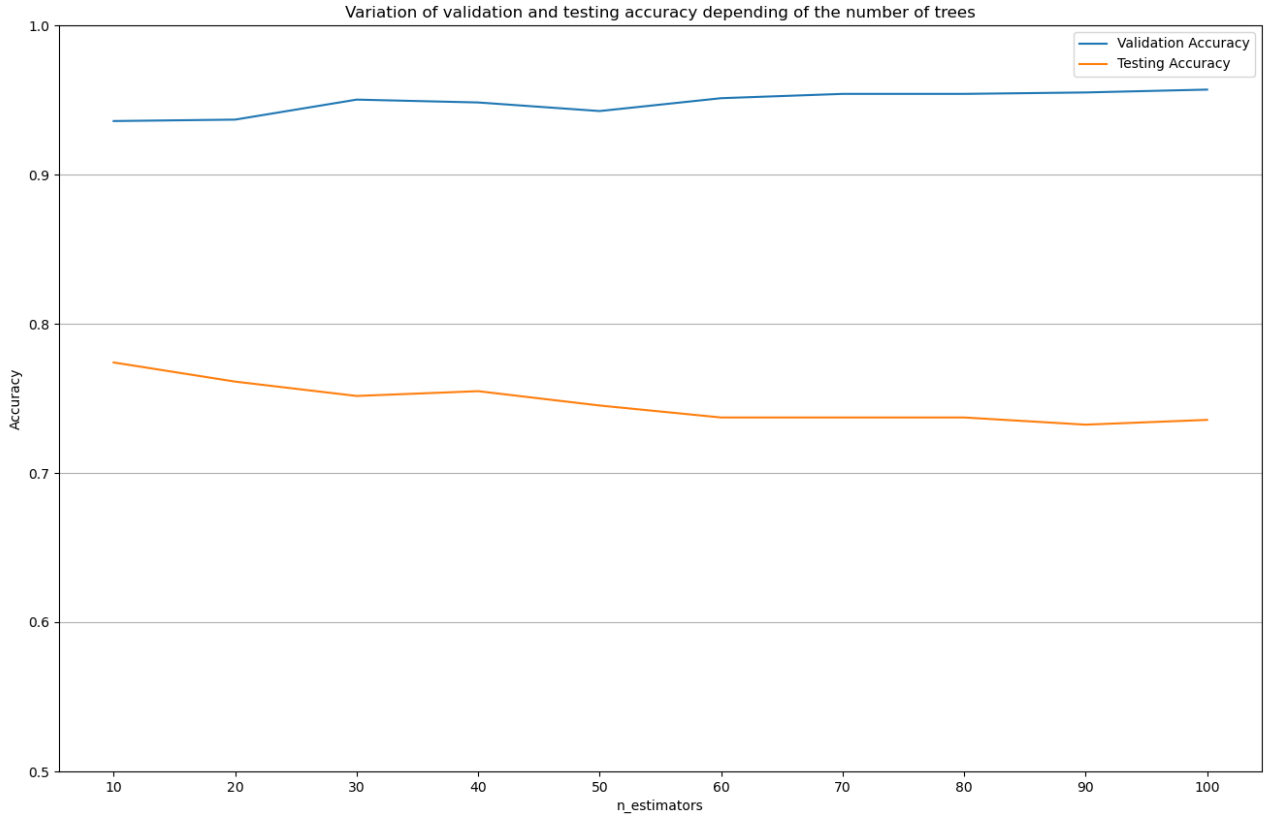


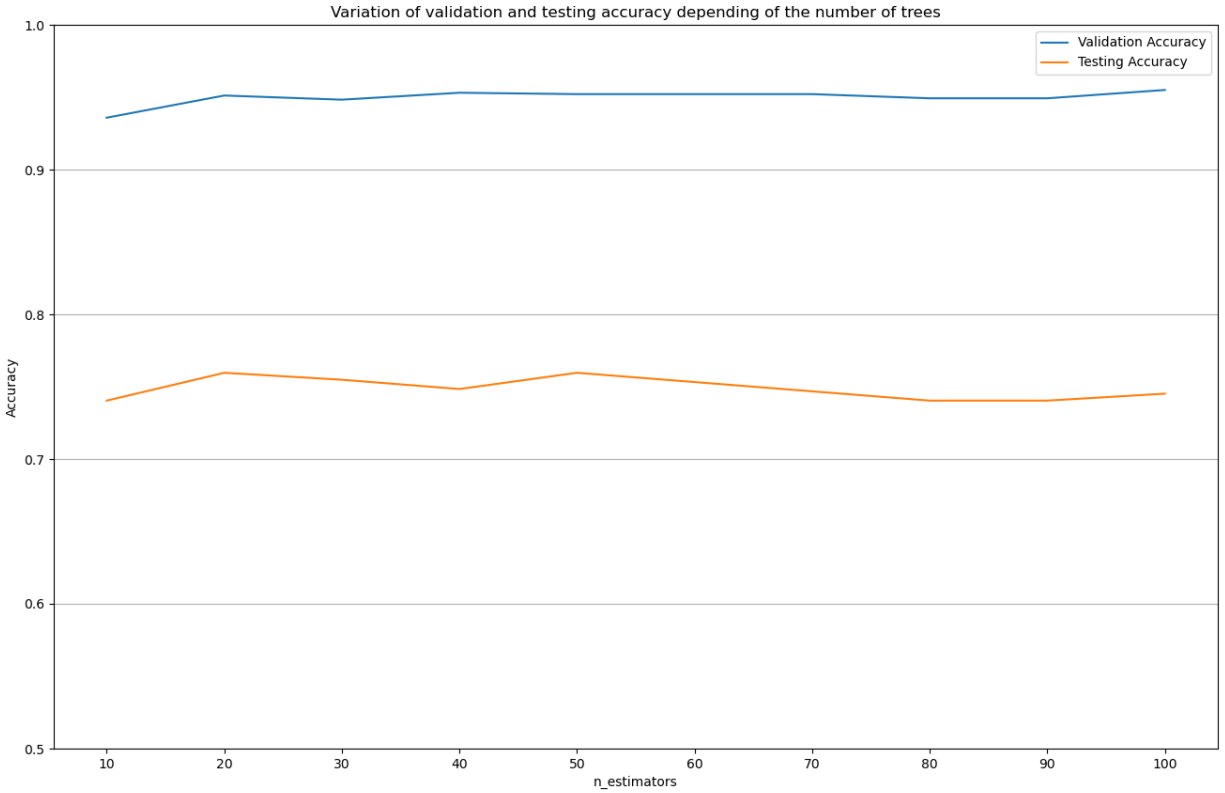


Les graphiques précédents ne montrent pas une amélioration notable de la précision en fonction de l'évolution du max_depth et min_samples_split, nous allons donc conserver les paramètres par défaut pour l'algorithme randomForest.

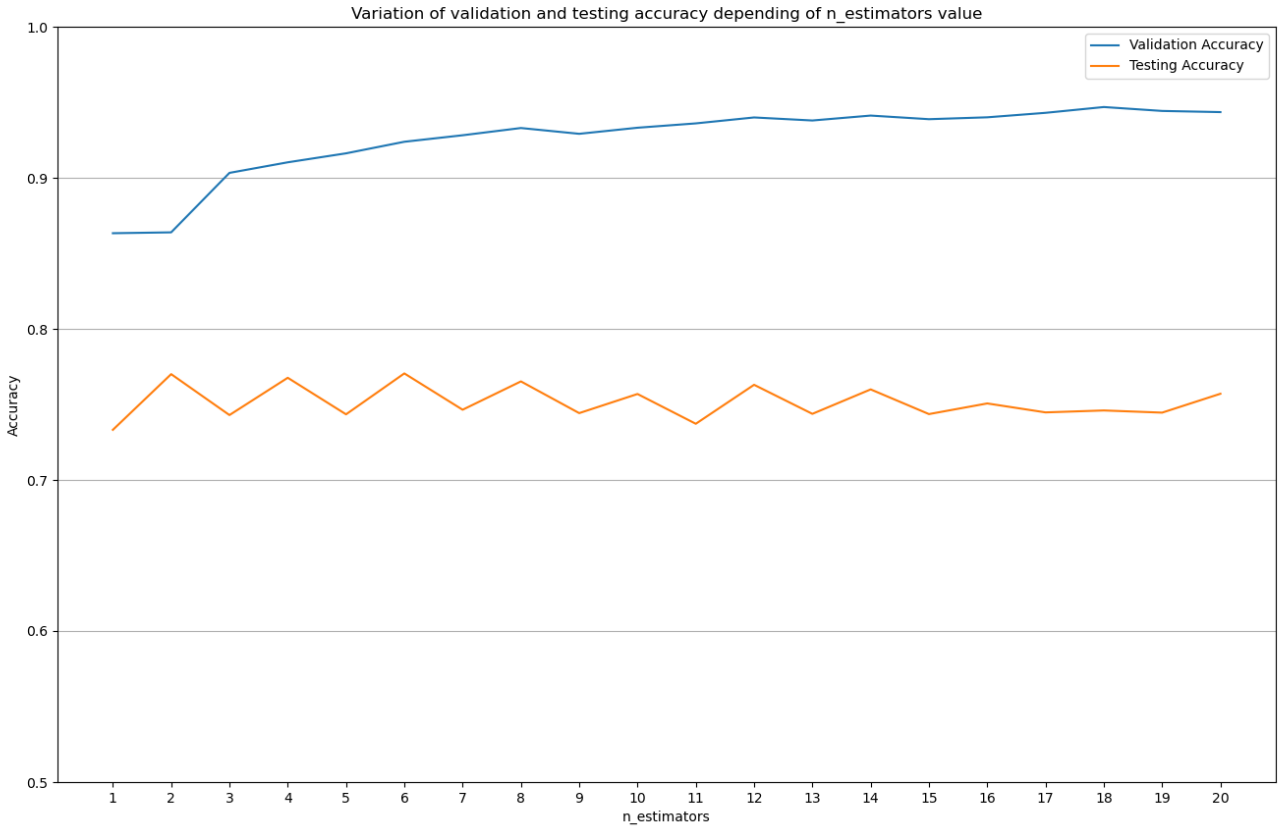
Nous allons à présent choisir une valeur de n_estimators pour notre modèle final :







Sur les graphiques, nous observons que la précision décroît légèrement lorsque l'on augmente n_estimator. Pour vérifier si cela est bien fondé, nous avons entraîné dix modèles pour chacune des valeurs de n_estimators entre 1 et 20, puis nous avons tracé leurs moyennes :



La précision moyenne maximale est obtenue pour n_estimator = 6, avec 0.7916 de précision en moyenne. La précision maximale 0.7917 a également été atteinte avec n_estimator = 6.

Pour les valeurs de n_estimators inférieur à 15, nous observons une précision significativement meilleure avec une valeur paire qu'avec une valeur impaire.

Avec une moyenne de seulement 10 valeurs, nous ne pouvons pas conclure que la valeur de n_estimators optimal est 6 pour notre cas d'utilisation. Cependant, nos résultats pointent clairement sur une valeur paire comprise entre 2 et 14. N'ayant pas réussi à obtenir un modèle plus précis que 0.7917, nous allons conserver le modèle créé avec n_estimator = 6 comme modèle de comparaison avec les autres algorithmes.

Conclusion :

En comparant les résultats obtenus de nos différents modèles, nous constatons que le Convolutional Neural Network est le modèle le plus efficace avec une précision obtenue de 0.8413, surpassant les précisions du Dense Neural Network à 0.7812 et de l'algorithme Random forest à 0.7917.

Ces résultats correspondent à ce que nous attendions, les CNN et les random forest étant beaucoup utilisés pour de la classification d'images.