Rapport du TP2 - CNN

Travail remis par
CAPONE, MICHAEL - CAPM17068409
FEDOROVA, OLGA - FEDO27619308
DUPRÉ, XAVIER - DUPX28029808

Travail remis à
Massardi, Jean

Dans le cadre du cours
INF5081 - Gestion et analyse de données

Le 17 décembre 2021
Université du Québec à Montréal (UQÀM)

Table des matières

1. Les ensembles de données

Pour réaliser l'entraînement des réseaux de neurones convolutifs, deux groupes d'images différents ont été utilisés. Ainsi, pour chaque architecture, l'ensemble de données utilisé est indiqué dans le champ *Dataset*. À noter que les deux datasets sont balancés pour chaque classe et contiennent donc le même nombre d'images par classe ou presque.

Champignons

Ceci correspond à l'ensemble d'images initial donné pour ce travail pratique. Il contient au total 1000 images de champignons avec une distribution équilibrée entre chaque classe, donc chaque classe a 100 images.

Champignons nettoyés

Dans l'ensemble de données de départ, 8 images étaient inadéquates pour faire la classification des champignons par leur espèce. Ces données ont été enlevées pour tenter d'améliorer l'entraînement de notre réseau. En effet, deux causes possibles qui peuvent provoquer de l'overfitting sont un dataset trop petit et des données contenant du bruit tel que des images mal étiquetées. Ainsi, ce dataset contient en tout 992 images. Voici la liste des chemins des images qui ont été supprimées:

- Fomes formentarius/CL2012PIC81751076.jpg;
- Mycena galericulata/DB2012PIC56037784.jpg;
- Mycena galericulata/DB2012PIC56360815.jpg;
- Mycena galericulata/DB2014PIC17608927.jpg;
- Mycena galericulata/EAT2011PIC54173740.jpg;
- Plicatura crispa/AA2017-9187386_Sye1hwdg9g.jpg;
- Plicatura crispa/BWP2017-9185943_B1Gwr_c8g.jpg;
- Plicatura crispa/BWP2017-9185943_B1h7SuqLe.jpg.

2. La librairie Keras

La librairie Keras a été utilisée pour réaliser ce travail pratique. Bien qu'un réseau de neurones convolutif commence par la couche Input, cette couche n'est pas explicitement définie lors de la construction du modèle avec Keras. Plutôt, on ajoute à la première couche cachée du réseau un paramètre et on lui passe les dimensions de l'image. Par exemple, pour une image en couleur de 64 pixels par 64 pixels, on utilise l'argument input_shape=(64,64,3). Pour alléger les tableaux, ce paramètre n'est pas spécifié dans la première couche du champ Architecture.

Dans la méthode d'entraînement du modèle, il existe un argument qui permet d'appliquer des rappels durant l'entraînement. Un de ces rappels permet de sauvegarder un modèle si la valeur d'un métrique surveillé atteint sa valeur maximale ou minimale. Pour ce projet, le rappel surveille la précision (*accuracy* en anglais pour éviter toute ambiguïté avec les métriques *precision* et *recall*) sur l'ensemble de validation et sauvegarde un nouveau modèle à chaque fois que la valeur obtenue pour une époque est supérieure à la dernière valeur maximale.

3. Les architectures

Cette section contient les étapes importantes de la construction de notre architecture dans le cadre de ce travail pratique. Les explications de la motivation pour les prochains changements sont détaillées après l'affichage des paramètres de notre réseau et des résultats. Plusieurs recherches et tentatives ont été effectuées pour déterminer les hyperparamètres à utiliser, l'architecture optimale du réseau et les paramètres d'augmentation des images dans le but d'améliorer la performance de notre modèle ainsi que de diminuer son surapprentissage. Pour plusieurs essais, les gains de performance étaient minimes malgré l'utilisation de techniques bien connues.

Architecture #1

Informations	Valeurs
Architecture	Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Flatten() Dense(units=10, activation='softmax')
Learning rate	1.0
Optimiseur	Adam
Batch	800
Epoch	5
Itération	1
Split	80-20
Taille de l'image	64x64
Mode de couleur	rgb
Augmentations	rescale=1.0/255.0
Dataset	Champignons (1000 images)
Vitesse d'apprentissage	5 sec/epoch. Temps total d'entraînement: 25 secondes
Training loss	2.8592
Training accuracy	0.1037
Training precision	0.5000
Training recall	0.0025
Validation loss	2.7323
Validation accuracy	0.1000
Validation precision	0.0000
Validation recall	0.0000

Pour cette architecture, on a obtenu une précision de 10%. Puisqu'il existe 10 classes dans l'ensemble d'images, la précision obtenue est égale au hasard pur. Avec 5 époques, 1 itération et un lot de 800 images, le réseau n'a reçu que 5 mises à jour des poids avec la rétropropagation, ce qui explique cette précision. Le réseau n'a pas eu assez d'entraînement pour modifier ses poids et ainsi pour être capable de généraliser sur l'ensemble de validation.

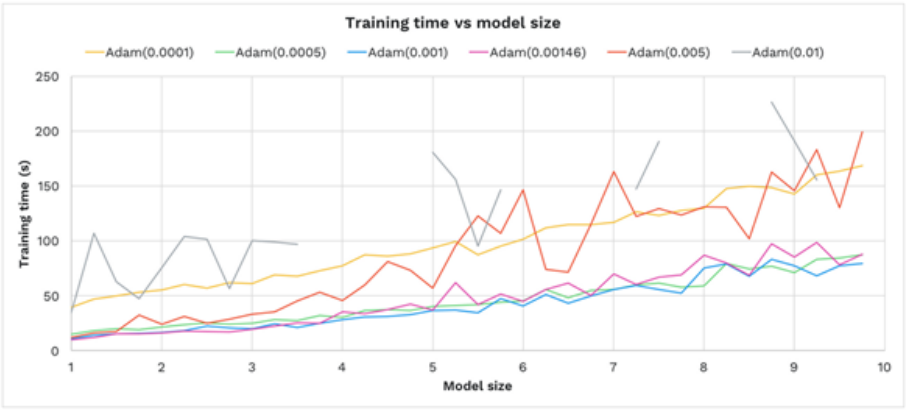
Dans la prochaine architecture, des couches de convolution avec plus de filtres et des couches de pooling seront ajoutées au réseau pour augmenter sa profondeur et sa largeur. Le réseau sera donc capable de se focaliser sur les caractéristiques de haut niveau des champignons dans les couches cachées, ce qui améliorera son pouvoir de généralisation. De plus, certains hyperparamètres seront aussi modifiés pour que le réseau soit capable de modifier les poids avec le bon taux d'apprentissage.

Le taux d'apprentissage actuel est trop élevé, ce qui fait en sorte que l'algorithme diverge et donne une solution sous-optimale : la descente du gradient fait des pas trop grands et n'arrive pas à converger vers un minimum local, mais plutôt passe d'un côté à l'autre de celui-ci. Alors, il faudra diminuer le taux d'apprentissage pour améliorer la recherche du minimum local.

À partir des recherches effectuées sur le taux d'apprentissage, l'équipe a décidé de choisir Adam comme optimiseur puisqu'il est très populaire. Nous avons aussi lu que la valeur optimale du taux d'apprentissage se situe entre 0.0005 et 0.001. Ayant testé de nombreuses valeurs, nous avons remarqué que les meilleures performances de notre modèle étaient obtenues avec une valeur de 0.0005.

Which learning rate performs best for different sizes of model?

Let's run the same experiment for multiple learning rates and see how training time responds to model size:



Failed runs are shown as missing points and disconnected lines

- Learning rates 0.0005, 0.001, 0.00146 performed best — these also performed best in the first experiment. We see here the same "sweet spot" band as in the first experiment.

https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2

Ensuite, le nombre d'époques devrait être beaucoup plus grand, puisque la valeur de 1 ne permet pas en ce moment d'entraîner suffisamment le réseau. D'ailleurs, en diminuant le taux d'apprentissage, il convient normalement d'augmenter le nombre d'époques afin de compenser les modifications des poids plus petites apportées à chaque mise à jour.

Enfin, la taille des lots sera plus petite pour l'architecture suivante. Puisque l'entraînement sur l'ensemble d'images peut demander beaucoup de mémoire, il est préférable de diviser les images en sous-groupes. Lorsque les dimensions des images seront plus grandes, il serait possible de dépasser la limite de RAM permise si on utilisait des lots de 800 images.

Architecture #2

Informations	Valeurs
Architecture	Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Flatten() Dense(units=10, activation='softmax')
Learning rate	0.0005
Optimiseur	Adam
Batch	32
Epoch	50
Itération	25
Split	80-20
Taille de l'image	64x64
Mode de couleur	rgb
Augmentations	rescale=1.0/255.0
Dataset	Champignons (1000 images)
Vitesse d'apprentissage	5 sec/epoch. Temps total d'entraînement: 250 secondes
Training loss	0.1663
Training accuracy	0.9837
Training precision	0.9948
Training recall	0.9588
Validation loss	3.4170
Validation accuracy	0.2900
Validation precision	0.3125
Validation recall	0.2250

En diminuant la valeur du taux d'apprentissage, en augmentant le nombre d'époques et en ajoutant plus de couches de convolution et de couches de pooling dans le réseau, on obtient une précision de 98.37% pour l'ensemble d'entraînement et 29.00% pour celui de validation. Ainsi, notre réseau est maintenant meilleur que le hasard pur pour prédire l'espèce d'un champignon à partir d'une image.

Cependant, on remarque qu'il existe une très grande différence entre la précision de l'ensemble d'entraînement et celle de validation. Cela signifie que notre réseau est overfit. En effet, après 50 époques, les poids du réseau ont été modifiés de sorte à parfaitement identifier les images d'entraînement ; par contre, lorsqu'on lui demande de classifier les images de validation, la précision baisse considérablement, ce qui témoigne de la faible capacité de généralisation du modèle actuel.

Pour la prochaine architecture, l'objectif sera de diminuer l'overfitting. Pour ce faire, nous allons appliquer plusieurs approches pour que la précision de l'ensemble d'entraînement se rapproche le plus possible de celle de l'ensemble de validation. Une technique à utiliser sera l'augmentation des données dans laquelle les images de l'ensemble d'entraînement recevront des transformations qui les feront varier. Ainsi, à chaque instance d'entraînement, les groupes d'images seront un peu différents par rapport aux instances précédentes. Les augmentations sont le retournement horizontal et vertical, la rotation et le zoom.

Une autre approche sera l'ajout de couches Dropout dans le réseau. Cette couche fait baisser l'overfitting en ignorant de manière aléatoire des neurones d'entrée durant une étape de l'entraînement. Le pourcentage de neurones ignorés sera mis à 25%. En effet, avec une valeur plus petite, le réseau aurait encore trop de surapprentissage, mais avec une valeur trop grande, le réseau serait incapable d'apprendre correctement. Ainsi, le réseau évite de faire un surapprentissage sur les données d'entraînement.

Finalement, il faudra nettoyer l'ensemble de données, puisque certaines images sont erronées, alors que d'autres sont trop bruitées. En effet, on retrouve une image de papillon et une image complètement noire dans les dossiers. Dans un autre dossier, des images sont des dessins de l'espèce de champignon et nous avons jugé qu'il est préférable de les enlever. Enfin, on devra enlever deux images ayant un homme au centre avec les champignons affichés sur le côté.

Architecture #3

Informations	Valeurs
Architecture	Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding = 'same') Dropout(0.25) MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same') Dropout(0.25) MaxPool2D(pool_size=(2, 2), strides=2) Flatten() Dense(units=10, activation='softmax')
Learning rate	0.0005
Optimiseur	Adam
Batch	32
Epoch	100
Itération	25
Split	80-20
Taille de l'image	64x64
Mode de couleur	rgb
Augmentations	rescale=1.0/255.0, horizontal_flip=True, vertical_flip=True, rotation_range=90, zoom_range=0.5
Dataset	Champignons nettoyés (992 images)
Vitesse d'apprentissage	6 sec/epoch. Temps total d'entraînement: 600 secondes
Training loss	1.6058
Training accuracy	0.4214
Training precision	0.6457
Training recall	0.1811
Validation loss	2.0169
Validation accuracy	0.2893
Validation precision	0.3636
Validation recall	0.0203

En ajoutant des paramètres d'augmentation de données et des couches de Dropout après les couches d'activation, la précision de l'ensemble d'entraînement a baissé à 42,14%, alors que la précision de l'ensemble de validation est restée sensiblement la même à 28,93%. Ces résultats sont plutôt satisfaisants, puisque cela signifie que les changements apportés au réseau ont fait diminuer l'overfitting de manière considérable. Il faut aussi noter que le nombre d'époques est passé de 50 à 100 puisque l'apprentissage des données d'entraînement prenait plus de temps avec les techniques employées.

Maintenant que le surentraînement est moins important, il faudra continuer à améliorer la précision du réseau. Pour la prochaine architecture, on élargira les couches de convolution en leur donnant beaucoup plus de filtres pour améliorer la capacité du réseau à s'entraîner. Effectivement, les couches de convolution enverront plus d'information aux couches suivantes. On augmentera aussi la profondeur du réseau en ajoutant une couche de convolution supplémentaire suivie d'une couche de pooling pour essayer d'améliorer la reconnaissance des caractéristiques des champignons.

Il sera nécessaire de diminuer le nombre de couches Dropout, puisque avoir trop de couches fait en sorte que l'entraînement est trop lent. En effet, avec les hyper paramètres choisis pour l'architecture, la précision sur l'ensemble d'entraînement était sous 50%, alors que l'objectif est d'avoir une valeur autour de 80%. Cette couche Dropout sera déplacée après la couche d'écrasement au lieu d'être après les couches de convolution.

Puisque la profondeur et la largeur seront plus grandes dans l'architecture suivante, il serait judicieux d'augmenter les dimensions des images pour augmenter leur quantité d'information. On va prendre des valeurs qui sont des puissances de 2 pour augmenter la vitesse de calcul. Alors, la taille des images ira de 64 par 64 pixels à 128 par 128 pixels. Les canaux de couleurs n'auront pas besoin d'être changés, puisqu'il aurait une perte d'information importante en utilisant des images en niveau de gris.

Architecture #4

Informations	Valeurs
Architecture	Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding = 'same') MaxPool2D(pool_size=(2, 2), strides=2) Flatten() Dropout(0.25) Dense(units=10, activation='softmax')
Learning rate	0.0001
Optimiseur	Adam
Batch	32
Epoch	100
Itération	25
Split	80-20
Taille de l'image	128x128
Mode de couleur	rgb
Augmentations	rescale=1.0/255.0, horizontal_flip=True, vertical_flip=True, rotation_range=90, zoom_range=0.5
Dataset	Champignons nettoyés (992 images)
Vitesse d'apprentissage	20 sec/epoch. Temps total d'entraînement: 2000 secondes
Training loss	1.3424
Training accuracy	0.5220
Training precision	0.7601
Training recall	0.3069
Validation loss	1.8686
Validation accuracy	0.3858
Validation precision	0.5067
Validation recall	0.1929

Avec les changements effectués sur l'architecture du réseau et sur les dimensions des images, la précision sur l'ensemble d'entraînement est de 52,2% et la précision sur l'ensemble de validation est de 38,58%. Il faut noter que le taux d'apprentissage a baissé par rapport à l'architecture précédente, puisque les résultats obtenus étaient moins satisfaisants avec la dernière valeur. Les valeurs des précisions ont augmenté par rapport au réseau précédent, mais les résultats ne sont pas encore parfaits.

Dans le dernier réseau, on ajoutera une couche de normalisation de lots après chaque couche de convolution. L'utilité de cette couche est de standardiser les entrées. L'implémentation de la normalisation de lots aura pour effet d'accélérer le processus d'entraînement et d'améliorer légèrement la performance du modèle. Il ne sera donc pas nécessaire d'augmenter le nombre d'époques ou d'ajuster le taux d'apprentissage du réseau pour obtenir de meilleurs résultats plus rapidement.

Certaines couches de convolution plus loin dans le réseau auront davantage de filtres pour améliorer la reconnaissance des attributs de haut niveau des images. Lors de tests, l'équipe a constaté que de donner 512 filtres à la quatrième couche de convolution au lieu de 256 augmente légèrement la précision du réseau. Toutefois, le nombre de couches de convolution et de pooling ne sera pas modifié, puisque la profondeur semble adéquate.

Aussi, pour améliorer la convergence, nous avons décidé d'utiliser le taux d'apprentissage adaptatif de l'optimiseur Adam. Cet ajout permet d'adapter la valeur du learning rate selon le stade de l'apprentissage plutôt que de toujours avoir la même valeur.

Enfin, des changements au niveau de l'augmentation des images auront lieu pour encore limiter le surapprentissage. Les images augmentées pour l'entraînement auront aléatoirement un effet de cisaillement et un décalage de luminosité. L'intervalle de rotation sera plus petit, puisque la valeur actuelle est trop élevée. On doit faire en sorte que les images générées restent réalistes. D'ailleurs, le nombre de lots devra diminuer aussi pour limiter l'utilisation de la mémoire.

Architecture #5

Informations	Valeurs
Architecture	Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding = 'same') BatchNormalization() MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same') BatchNormalization() MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'same') BatchNormalization() MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding = 'same') BatchNormalization() MaxPool2D(pool_size=(2, 2), strides=2) Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding = 'same') BatchNormalization() MaxPool2D(pool_size=(2, 2), strides=2) Flatten() Dropout(0.25) Dense(units=10, activation='softmax')
Learning rate	0.0001
Optimiseur	Adam (beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
Batch	16
Epoch	100
Itération	50
Split	80-20
Taille de l'image	128x128
Mode de couleur	rgb
Augmentations	rescale=1.0/255.0, horizontal_flip=True, vertical_flip=True, rotation_range=45, brightness_range=[0.75,1.25], shear_range=0.25, zoom_range=0.5
Dataset	Champignons nettoyés (992 images)

Vitesse d’apprentissage	27 sec/epoch. Temps total d'entraînement: 2700 secondes
Training loss	0.8528
Training accuracy	0.7082
Training precision	0.7580
Training recall	0.6541
Validation loss	2.2898
Validation accuracy	0.4416
Validation precision	0.4812
Validation recall	0.3909

4. Autres expérimentations

Cette section contient d'autres techniques et tests qui ont été essayés mais qui n'ont pas donné de résultats significatifs pour se retrouver dans une architecture.

Régularisation L2

La méthode de régularisation L2 a été utilisée dans plusieurs architectures et avec plusieurs valeurs en hyperparamètre sans jamais donner de gains en précision. Bien que cette méthode soit connue pour aider avec le problème d'overfitting, ce ne fut pas le cas pour nous.

Fonction d'activation

Bien que nos architectures utilisent toutes la fonction d'activation RELU, nous avons expérimenté avec d'autres, telle que la fonction sigmoïde. Pour une même architecture, si on changeait la fonction d'activation, on observait des variations dans les performances. Toutefois, les meilleurs résultats étaient obtenus avec la fonction RELU.

Augmentation du nombre d'images

Une des raisons du surapprentissage est l'utilisation d'un ensemble de données trop petit pour faire l'entraînement. Pour tenter de résoudre ce problème, deux répertoires d'images ont été créés avec l'aide de Google (4000 images) et de Bing (12000 images). La précision n'a pas été grandement améliorée et le surapprentissage n'a pas été corrigé. En effet, nous avons obtenu seulement 1-2% de plus de précision pour certaines architectures, mais pas plus. Nous avons donc décidé de conserver l'ensemble de données original.

Images en noir et blanc

Nous avons essayé des architectures à l'aide d'images en noir et blanc sans toutefois obtenir de meilleurs résultats. Cependant, étant donné que la matrice pour une image en noir et blanc est trois fois plus petite, le temps d'apprentissage était beaucoup plus rapide.

5. Bibliographie

Voici les sources les plus importantes que l'équipe a utilisées pour les recherches.

1- https://towardsdatascience.com/

2- https://machinelearningmastery.com

3- https://medium.com

4- https://deeplizard.com