TP2 - CNN

Automne 2021 - INF 5081 - UQAM

Jean Massardi

Auteurs:

Pascale Lavoie	LAVP22627602
Abi Ayad Fethi Bey	ABIF10108204

Objectifs :

Le but de ce TP est de réaliser un système capable de reconnaître des champignons, de trouver leur espèce et de déterminer s' ils sont comestibles ou pas.

Le but pédagogique de ce TP est de faire réaliser un réseau de neurones type CNN sans utiliser un réseau pré-entraîné. Les étudiants devront déterminer l'architecture ainsi que les hyperparamètres à donner à leur réseau.

Résumé des modèles entraînés:

Modèle	**1**	**2**	**3**	**4**	**5**
Référence	model_ v1_0	model_ v1_2	model_ v2_7	model_ v4_0	model_ v4_128
Résumé	Modèle de base	Amélioration des filtres	Augmentation dataset avec flow from directory	Agmentation dataset avec flow form directory + image	Augmentation de la grosseur de l'image
Architecture du réseau	1 couche Conv+ pool Filtre: 128 Sortie avec: Dense Pour les détails voir annexe	4 couches Conv+ Pool+ dropout Filtres: 24,128,256,768 Sortie avec: Dense, Dropout, Dense Pour les détails voir annexe	4 couches Conv+ Pool+ dropout Filtres: 24,128,256,768 Sortie avec: Dense, Dropout, Dense Pour les détails voir annexe	4 couches Conv+ Pool+ dropout Filtres: 24,128,256,768 Sortie avec: Dense, Dropout, Dense Pour les détails voir annexe	4 couches Conv+ Pool+ dropout Filtres: 24,128,256,768 Sortie avec: Dense, Dropout, Dense Pour les détails voir annexe
Grandeur image	64 x 64 x 3	64 x 64 x 3	64 x 64 x 3	64 x 64 x 3	128 x 128 x 3
Modification du dataset	Non Voir détails en annexe	Non Voir détails en annexe	Oui avec ImageDataGenerator Voir détails en annexe	Oui - avec ImageDataGenerator et ajout d'images Voir détails en annexe	Oui - avec ImageDataGenerator et ajout d'images Voir détails en annexe
Taille dataset	800, 200	800, 200	800, 200	2986, 741	2986, 741
Répartition du dataset	80% entraînement / 20% validation	80% entraînement / 20% validation	80% entraînement / 20% validation	80% entraînement / 20% validation	80% entraînement / 20% validation

Learning rate	Par défaut avec adam (0.001)	Par défaut avec adam (0.001)	Par défaut avec adam (0.001)	Par défaut avec adam (0.001)	Par défaut avec adam (0.001)
Optimiseur	adam	adam	adam	adam	adam
Batch	800	128	800	128	128
Epochs	5	200	800	1000	800
Itérations	1	6.25	1	23.3	23.3
Précision	0.20	0.2800	0.4099	0.4615	0.5452
Recall	0.20	0.28	0.43	0.48	0.55
temps /epoch	12 s	11 s	11 s	41 s	85 s
temps total	60 s	37 min	2 h 27 min	11 h 23 min	18 h 53 min
Analyse	Voir annexe 1	Voir annexe 2	Voir annexe 3	Voir annexe 4	Voir annexe 5

Annexe 1 - Réseau #1

Ce modèle est le modèle de base demandé dans l'énoncé du TP
Reference model_v1_0

Modification du dataset:

En théorie, il n'y a pas de modification du dataset. ImageDataGenerator et flow_from_directory sont utilisés. flow_from_directory utilise un batch_size par défaut qui est de 32.

Architecture:

Construction du modèle: (pour les conv2D activation utilisée = relu)

```
Conv2D(128, kernel_size=(3, 3)) + MaxPooling2D(pool_size=(2, 2)
Flatten()
Dense(10, activation='softmax')
```

Analyse:

Dans la conception du modèle on a utilisé RELU comme fonction d'activation sauf la dernière couche à laquelle on utilise la fonction d'activation softmax puisque nous avons plusieurs classes.

Dans la compilation du modèle on a ajouté l'information de précision.

Observations faites lors de nos entraînements:

1. Pendant l'entraînement d'un modèle, nous obtenons une valeur de précision d'entraînement. Lorsque celle-ci n'atteint pas 0.9 ou plus, la précision du modèle peut être améliorée en augmentant le nombre d'epoch. A la dernière Epoch, le résultat de l'entraînement nous montre une précision d'entraînement de 0.8188 et une précision de 0.1949. Il est donc possible d'aller chercher un peu plus de précision en augmentant le nombre d'epoch.
2. Pour un même modèle et avec les mêmes paramètres d'entraînement, il peut y avoir des variations dans le précision de 0.05.
3. La précision peut parfois rester à 0.1 pendant environ 10 epoch avant de commencer à augmenter.

Note: Lors de l'entraînement du modèle, Keras nous donne des valeurs de "accuracy" et de "val_accuracy". Le val_accuracy est la valeur de précision résultant des essais réalisés avec le dataset de validation. Pour le reste du rapport on utilisera précision pour parler du "val_accuracy" et de précision d'entraînement pour le "accuracy" retourné par Keras.

Annexe 2 - Réseau #2

Ce modèle est la recherche de filtres afin d'obtenir une précision optimale sans augmenter le dataset.

Reference: model_v1_2

Modification du dataset:

En théorie il n'y a aucune modification du dataset, par contre en utilisant flow_from_directory, ceci créer un batch_size de 32 par défaut que nous avons conservés pour le dataset d'entraînement et le dataset de validation.

Architecture: pour les conv2D activation utilisée = relu

- Conv2D(filters= 24, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=128, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=256, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=768, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2)))
- Flatten())
- Dense(128, activation='relu')
- Dropout(0.50)
- Dense(10,activation='softmax')

Analyse:

1. Nous avons ajouté des couches de filtres pour augmenter la précision, et des couches de dropout pour diminuer "l'overfitting". Des couches de pooling sont aussi utilisées à chaque couche de filtre afin de conserver moins d'information et de pouvoir réduire le temps d'entraînement.

   Observations faites lors de nos entraînements:

   2. Il est possible d'ajouter des couches de pooling lorsqu'on augmente la grosseur de l'image. Pour une image de 64 x 64, le nombre de pooling dont le "pool_size" est de 2,2 se limite à 4.
   3. Lors d'ajout de filtres la précision diminue, pour compenser, on peut alors augmenter le nombre d'epoch.
   4. Des essais avec un callBack de "EarlyStop" ont été essayés afin de maximiser le temps d'entraînement. Cependant ces essais n'ont pas donné l'effet escompté; la précision obtenue pendant l'entraînement ne montait pas assez haut, et ceci même en augmentant la valeur de "patience". Possiblement qu'il y avait trop d'overfitting dans le modèle pour fonctionner adéquatement. Ces essais n'ont pas été conservés.

Annexe 3 - Réseau #3 - Augmentation du dataset avec ImageDataGenerator

Reference: model_v2_7

Modification du dataset:

Modifications avec ImageDataGenerator:

fill_mode='nearest',
shear_range=0.2,
horizontal_flip=True,
rotation_range=40,
brightness_range=[0.5,1.5],
width_shift_range=0.2,
height_shift_range=0.2

Paramètres de flow_from_directory:

batch_size = 512 (pour le dataset d'entraînement)
batch_size = 512 (pour le dataset de test)

Architecture: pour les conv2D activation utilisée = relu

- Conv2D(filters= 24, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=128, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=256, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=768, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2)))
- Flatten())
- Dense(128, activation='relu')
- Dropout(0.50)
- Dense(10,activation='softmax')

Analyse:

En augmentant de façon artificielle le dataset à l'aide de ImageDataGenerator de Keras, on augmente la précision de notre modèle de 0.13 en diminuant l"overfitting".

Observations faites pour arriver à ce modèle:

Dans l'augmentation du dataset avec ImageDataGenerator, nous avions initialement le paramètre "zoom_range=0.2", puisque celui-ci déforme énormément les images de base. Il a été enlevé pour ne pas créer de l'overfitting au modèle.

Par défaut, l'"optimiser" Adam utilise un learning rate de 0.001. Des tests ont été fait à ce sujet et augmenter sa valeur nous a donné de mauvais résultats. La diminution de la valeur augmentait considérablement le temps d'apprentissage. Le learning rate par défaut a donc été conservé.

Annexe 4 - Réseau #4

Reference: model_v4_0

Modification du dataset:

Modifications avec ImageDataGenerator:

fill_mode='nearest',
shear_range=0.2,
horizontal_flip=True,
rotation_range=40,
brightness_range=[0.5,1.5],
width_shift_range=0.2,
height_shift_range=0.2

Paramètres de flow_from_directory:

batch_size de 256 pour le dataset d'entraînement
batch_size de 1 pour le dataset de validation

Ajout des images du dataset original:

A partir du dataset original:
https://www.kaggle.com/c/fungi-challenge-fgvc-2018/overview
Chaque sous-répertoire(classe) contient entre 309 et 443 images. Chaque classe est donc assez bien balancée les unes par rapport aux autres.

Architecture: pour les conv2D activation utilisée = relu

- Conv2D(filters= 24, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=128, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=256, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=768, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2)))
- Flatten())
- Dense(128, activation='relu')
- Dropout(0.50)
- Dense(10,activation='softmax')

Analyse:

Avec l'ajout de vraies images dans le dataset augmente la précision de 0.05 en diminuant l'"overfitting" tout comme l'a fait l'ajout d'images avec le ImageDataGenerator. Celui-ci a l'avantage d'ajouter des images qui risquent de ressembler davantage à des images que les utilisateurs fourniront au modèle.

Un batch_size plus petit a tendance à donner de meilleur résultat (ref: **https://stats.stackexchange.com/questions/164876/what-is-the-trade-off-between-batch-size-and-number-of-iterations-to-train-a-neu**). L'utilisation d'un batch_size de 128 a donc été privilégiée dans se modèle contrairement à un batch_size de 800 pour les modèles 1 et 3.

Observations faites pour arriver à ce modèle:

On a fait un modèle basé sur ce blog:
https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
En gros le dataset est utilisé au complet pour l'entraînement et la validation. Le modèle a atteint avec une seule itération et 300 epochs une précision de 0.7778. En ajoutant plus d'epochs ou en ajoutant une autre itération le "val_accuracy" pourrait encore augmenter de façon à obtenir une meilleure précision. Par contre ce modèle a été mis de côté car il utilise le même dataset pour la validation et le test. Il est fort sujet à de l'over fitting.

Annexe 5 - Réseau #5

Reference: model_v4_128

Modification du dataset:

Modifications avec ImageDataGenerator:

fill_mode='nearest',
shear_range=0.2,
horizontal_flip=True,
rotation_range=40,
brightness_range=[0.5,1.5],
width_shift_range=0.2,
height_shift_range=0.2

Paramètres de flow_from_directory:

batch_size de 256 pour le dataset d'entraînement
batch_size de 1 pour le dataset de validation

Ajout des images du dataset original:

A partir du dataset original:
https://www.kaggle.com/c/fungi-challenge-fgvc-2018/overview
Chaque sous-répertoire(classe) contient entre 309 et 443 images. Chaque classe est donc assez bien balancée les unes par rapport aux autres.

Architecture: pour les conv2D activation utilisée = relu

- Conv2D(filters= 24, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=128, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=256, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2))) + Dropout(0.25))
- Conv2D(filters=768, kernel_size=(3,3) + MaxPooling2D(pool_size=(2,2)))
- Flatten())
- Dense(128, activation='relu')
- Dropout(0.50)
- Dense(10,activation='softmax')

Analyse:

L'augmentation de la grosseur de l'image à 128 * 128 au lieu de 64 * 64 qui fournit au modèle 4 fois plus d'informations. Ceci augmente la précision de 0.08. Ce modèle est aussi beaucoup plus lent à entraîner.

Observations faites pour arriver à ce modèle:

Notre modèle actuel utilise moins de filtres en entrées et plus de filtres en sorties. D'autres tests ont été faits sur un modèle moins long à entraîner et ceux-ci démontraient que les filtres avec les valeurs inversés (768,256,128,24) donnait de moins bons résultats mais que des filtres de même valeurs: 289, 289, 289, 289 au lieu de 24, 128,256,768, donnaient de meilleurs résultats. Noter que le résultat de 289 x 4 est la même que 24+128+256+768. Par contre au lancement du modèle, avec le dataset augmenté, le temps d'exécution était trop long, ce modèle a été abandonné.

Références:

https://stats.stackexchange.com/questions/164876/what-is-the-trade-off-between-batch-size-and-number-of-iterations-to-train-a-neu, consulté le 8 décembre 2021

Table 2: Performance of small-batch (SB) and large-batch (LB) variants of ADAM on the 6 networks listed in Table 1

	Training Accuracy		Testing Accuracy	
Network Name	SB	LB	SB	LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$99.99\% \pm 0.04\%$	$99.99 \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$