Deep Learning:
La reconnaissance des Champignons

Bason Valentin
Dahon Doryan
Dary Jean-Leo
Gouneau Joceran
Pigneux Alice
M1

Département Sciences du Numérique - Deuxième année

2020-2021

Contents

1 Introduction

1.1 À propos du projet

Reconnaisance de champignons par un algorithme de deep-learning

Le groupe de projet est constitué de 5 personnes :

- Bason Valentin
- Dahon Doryan
- Dary Jean-Léo
- Gouneau Joceran
- Pigneux Alice

Voici le lien github de la base données du projet:
https://github.com/ddahon/mushroom-neural-network
et le lien googleColab du projet:
https://colab.research.google.com/drive/1nySeYNYUrxiQQa-X5OePyCfifyRigU8YscrollTo=djtaTZG3xoXq
Pour le reseau parralele (bonus dans notre projet) :
https://colab.research.google.com/drive/12iEqBvDi0uj9g0$_p$78$msN6XxS0v1eZovscrollTo = LIWwVMHZ4ars$

1.2 Description du sujet

Le projet consiste à classifier les 10 variétés de champignons suivantes :

Figure 1: Photos des 10 variétés de champignons

1.3 Constitution de la base de données

1.3.1 Acquisition et annotation des données

Les données ont été acquise grâce à un script Python (*src/scraping.py*) qui récupère les résultats du moteur de recherche Ecosia (qui utilise le moteur Bing). Pour cela le script fait 2 choses :

1. Ouverture du driver Mozilla *geckodriver* avec l'url correspondant à la recherche d'image (par exemple $https://www.ecosia.org/images?q=boletus\%20edulis$) puis attente que l'utilisateur aie scrollé assez pour faire apparaître suffisamment d'images.
   Une fois la quantité désirée atteinte, on parse le HTML pour récupérer les urls avec la librairie $BeautifulSoup$.

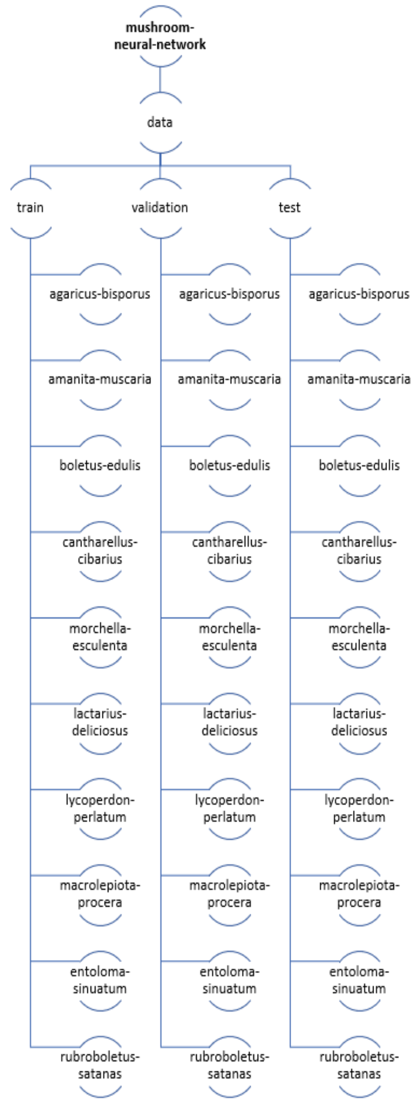
2. Téléchargement des images à partir des urls, les images sont stockées dans *data/nom-espece/nom-espece-numero*.

Les images sont donc annotées suivant le dossier dans lequel elles se trouvent.
Les recherches ont été effectuées avec le nom scientifique des espèces pour obtenir les résultats dans toutes les langues. Par exemple au lieu de rechercher "champignon de paris", on cherche "*Agaricus bisporus*".

Il a ensuite fallu vérifier les données à la main pour éliminer les doublons et les mauvaise images (champignons en bocaux ou cuisinés, image comportant plusieurs espèces différentes...)

1.3.2 Partitionnement des données

Pour cela nous allons utiliser une base de données de 2000 images, réparties en 1000 images d'apprentissage, 500 images de validation, et 500 images de test. Compte-tenu de la variabilité possible des représentations des champignons, cette base de données est d'une taille assez réduite et le problème est complexe.Nous avons réparti les images dans 3 ensembles (et donc 3 dossiers): *train* (3 images par classe), *validation* (1 image par classe) et *test* (1 image par classe). Chacun de ces dossiers comporte un sous-dossier par classe, qui contient les images correspondantes. L'arborescence est résumée sur l'image ci-dessus.



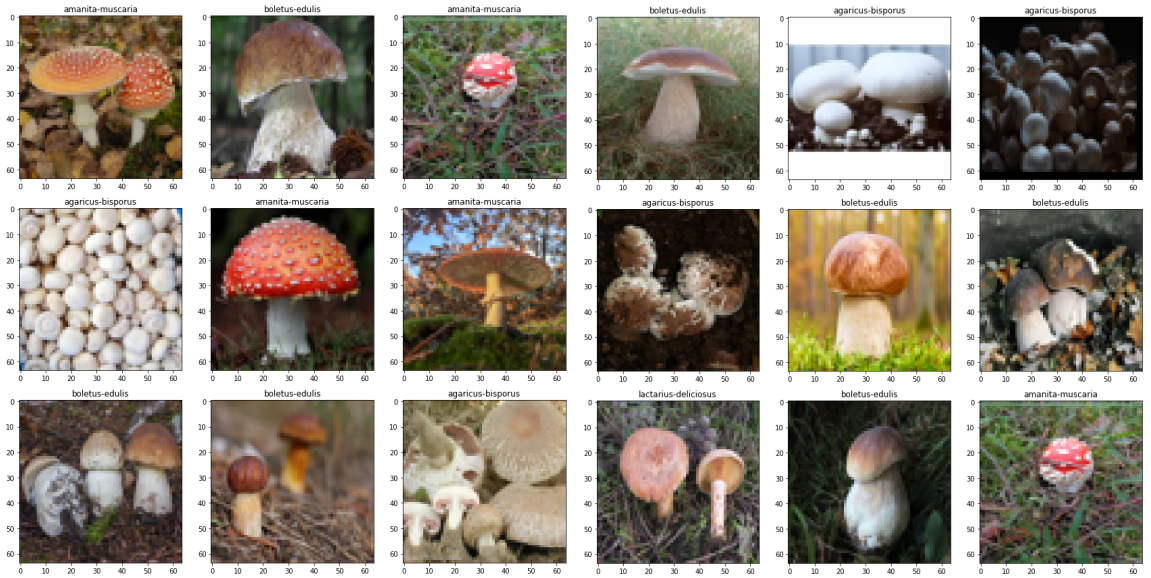
1.3.3 Notre pronostic

Le probleme est assez complexe. En effet, nous avons plusieurs classes avec une base de donnée réduite. D'autant plus que certain champignons se ressemblent beaucoup. (Par exemple entre le Lycoperdon Perlatum et le Macrolepiota Procera). Notre équipe pense que notre réseau ne sera pas optimal au vu de la complexité du problème (surtout si l'on veut reconnaitre de plus en plus de champignons). Mais nous comptons utiliser de l'augmentation de données pour pallier a ce problème.

1.4 Script de chargement des données

Nous avons mis notre script sur google Colab a des fins purement pratique par rapport a l'utilisation de GitHub. Voici le lien: Script Chargement des Donnnées

1.4.1 Aperçu de la base de données




lactarius-deliciosus
lycoperdon-perlatum
macrolepiota-procera
macrolepiota-procera
lactarius-deliciosus
morchella-esculenta
lycoperdon-perlatum
morchella-esculenta
lycoperdon-perlatum
lycoperdon-perlatum
morchella-esculenta
lycoperdon-perlatum
lycoperdon-perlatum
morchella-esculenta
macrolepiota-procera
macrolepiota-procera
macrolepiota-procera
macrolepiota-procera

2 Résolution du problème

La résolution du problème s'est faites en plusieurs étapes de complexité croissante afin de détecter au plus tôt des anomalies dans la base de données, des erreurs dans le chargement de celle-ci, ou divers corrections à apporter à tout ce qui est extérieur au modèle avant de déployer un réseau complexe.

2.1 Chargement des données

```
!git clone https://github.com/ddahon/mushroom-neural-network.git
path = "./mushroom-neural-network/data/"
mushrooms_names= ["agaricus-bisporus",
                  "amanita-muscaria",
                  "boletus-edulis",
                  "cantharellus-cibarius",
                  "entoloma-sinuatum",
                  "lactarius-deliciosus",
                  "lycoperdon-perlatum",
                  "macrolepiota-procera",
                  "morchella-esculenta",
                  "rubroboletus-satanas"]
```

Les données sont chargées depuis le git, puis, pour chacun des datasets `train`, `validation` et `test`, on effectue le traitement suivant : (ici uniquement pour `train` mais la procédure est la même pour les deux autres datasets)

1. Copie de toutes les images contenues dans chacun des sous-dossiers de `path/train` correspondant à chaque classe dans un nouveau sous dossier `path/train/train`.

```
#Images d'entrainement
train_filenames= []
for name in mushrooms_names:
  train_filenames.append(os.listdir(path + "train/" + name))

if not os.path.exists(path + "train/train"):
  os.mkdir(path + "train/train")

path_train = path + "train/"
for i in range(10):
for filename in train_filenames[i]:
    img = Image.open(path_train+mushrooms_names[i]+"/"+filename).convert('RGB')
    img = img.resize((image_size, image_size))
    img.save(path_train+"train/"+filename)
```

   Les images sont également redimensionnées en `(150, 150)` car, la plupart étant trop lourdes, les temps d'apprentissages des réseaux les plus simples étaient excessivement longs pour un dataset pourtant réduit.

2. Établissement des catégories associées aux données contenues dans `path/train/train` à partir du nom de chaque image.

```
train_filenames = os.listdir(path + "train/train")
train_categories=[]
for filename in train_filenames:
    category = filename.split('_')[0]
    train_categories.append(mushrooms_names.index(category))
```

3. Création des `dataframe` et `datagen` associés, avec normalisation (`rescale=1./255`) et taille de batch égale à `batch_size = 16`.

```
#Images d'entrainement
train_df = pd.DataFrame({
    'filename': train_filenames,
    'category': train_categories
})

train_datagen=ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_dataframe(
    train_df,
    path + "train/train/",
    x_col='filename',
    y_col='category',
    target_size=(image_size,image_size),
    class_mode='categorical',
    batch_size=batch_size
)
```

2.2 Premier réseau convolutif de base

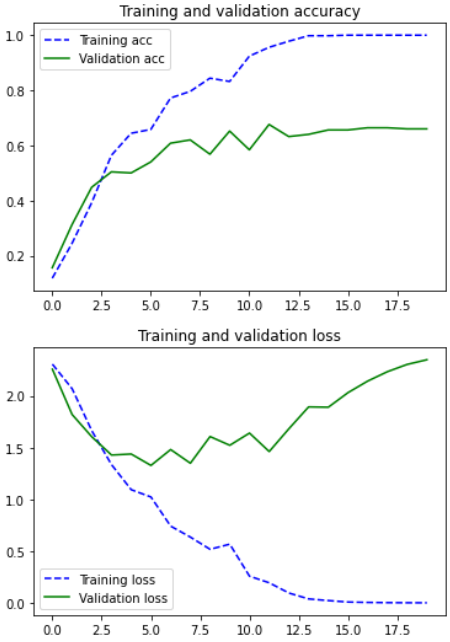
Layer (type)	Output Shape
Conv2D(filters=32, kernel_size=3, activation='relu')	(148, 148, 32)
MaxPooling(pool_size=(2,2)	(74, 74, 32)
Conv2D(filters=64, kernel_size=3, activation='relu')	(72, 72, 64)
MaxPooling(pool_size=(2,2)	(36, 36, 64)
Conv2D(filters=96, kernel_size=3, activation='relu')	(34, 34, 96)
MaxPooling(pool_size=(2,2)	(17, 17, 96)
Conv2D(filters=128, kernel_size=3, activation='relu')	(15, 15, 128)
MaxPooling(pool_size=(2,2)	(7, 7, 128)
Flatten()	(6272)
Dense(512, activation='relu')	(512)

```
Dense(10, activation='softmax')                                (10)
=================================================================================
```

Ci-dessus l'architecture du premier réseau basique, aucune régularisation n'a été implémentée lors de son entraînement de 20 `epoch`.

```
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(lr=3e-4),
              metrics=['acc'])
history = model.fit(train_generator, validation_data=validation_generator, epochs
    =20)
```

C'est avec l'entraînement de ce réseau que nous avons remarqué qu'il fallait redimmensionner les images dès leur chargement pour accélérer l'apprentissage. Voici les résultats de ce réseaux :



Entraînement du premier reseau

Le réseau apprend très bien les images d'apprentissage, ce qui est déjà un excellent début ; cependant malgré un apprentissage du réseau sur l'entièreté du dataset `train` le sur-apprentissage est excessif, il faut donc régler ce problème.

2.3 Augmentation de la base de données

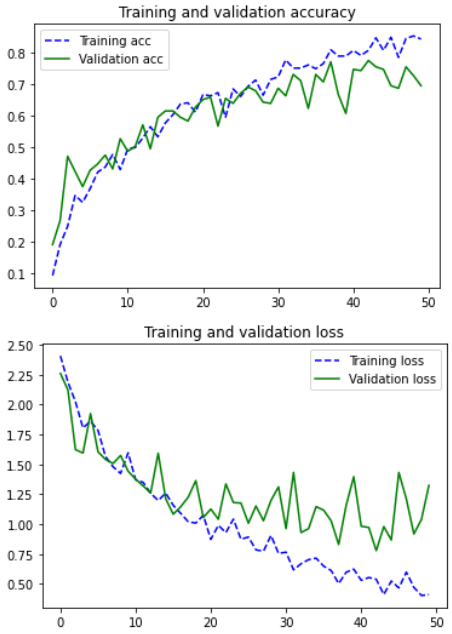
Une première manière pour amortir ce sur-apprentissage est d'augmenter la base de données, cette augmentation a été faite en définissant un nouveau `datagen` comme suit :

```
train_datagen_2 = ImageDataGenerator(
    rotation_range=40,
    rescale=1./255,
```

```
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)

train_generator_augmented = train_datagen_2.flow_from_dataframe(
    train_df,
    path + 'train/train/',
    x_col='filename',
    y_col='category',
    target_size=(image_size,image_size),
    class_mode='categorical',
    batch_size=batch_size
)
```

L'entraînement du même réseau que précédemment sur ce nouveau `datagen` et sur cette fois-ci 40 `epochs` donne les résultats sur la figure ci-dessous.



Entraînement du premier reseau sur les données augmentées

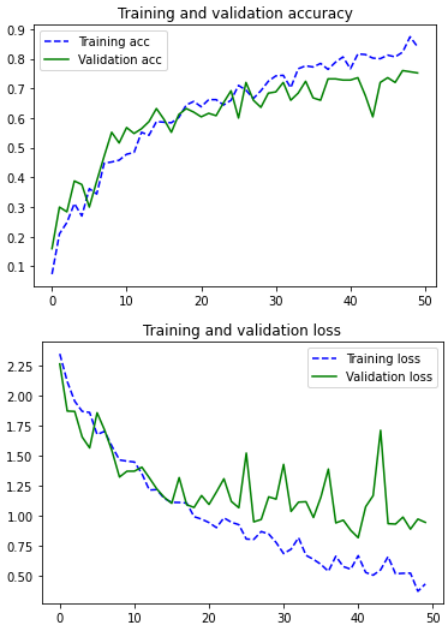
Le sur-apprentissage est bien amoindri, et au vu de la pente des courbes le réseau a encore de la marge d'apprentissage, même s'il faudrait sûrement ajouter une régularisation de type L_1 ou L_2 pour éviter un trop grand écart entre les résultats sur `train` et `validation` plus tard dans l'entraînement.

2.4 Implémentation d'un réseau plus complexe

Avant d'aller plus loins nous voulions évaluer les résultats d'un réseau plus complexe, afin de vérifier s'il valait mieux continuer avec une architecture différente. Voici alors l'architecture du réseau que nous testons, largement inspiré de la première moitié du U-net :

Layer (type)	Output Shape
Conv2D(filters=32, kernel_size=3, activation='relu')	(148, 148, 32)
Conv2D(64, 3, activation='relu', padding = 'same')	(148, 148, 64)
MaxPooling(pool_size=(2, 2)	(74, 74, 64)
Conv2D(128, 3, activation='relu', padding = 'same')	(74, 74, 128)
Conv2D(128, 3, activation='relu', padding = 'same')	(74, 74, 128)
MaxPooling(pool_size=(2, 2))	(37, 37, 128)
Conv2D(256, 3, activation='relu', padding = 'same')	(37, 37, 256)
Conv2D(256, 3, activation='relu', padding = 'same')	(37, 37, 256)
MaxPooling(pool_size=(2, 2))	(18, 18, 256)
Conv2D(512, 3, activation='relu', padding = 'same')	(18, 18, 512)
Conv2D(512, 3, activation='relu', padding = 'same')	(18, 18, 512)
MaxPooling(pool_size=(2, 2))	(9, 9, 512)
Flatten()	(41472)
Dense(512, activation = 'relu')	(512)
Dense(512, activation = 'relu')	(512)
Dense(10, activation = 'softmax')	(10)

Entraîné sur les données augmentées sur 40 `epochs` comme précédemment on obtients les courbes d'évolution suivantes :
Les résultats sont similaires à ceux du réseau précédent, mais légèrement plus encourageants.



Entraînement du deuxième reseau sur les données augmentées

2.5 Transfer-Learning et Fine-Tuning

La dernière approche que nous implémentons afin d'optimiser au maximum les résultats est celle du Transfer-Learning, qui permet d'avoir de bien meilleurs résultats qu'avec un réseau construit et entrainé uniquement sur cette base de données réduite. L'architecture du réseau est la suivante (avec une régularisation L_2 sur les deux couches denses à la fin avec `lambda = .08`) :

```
_________________________________________________________________________________________________
Layer (type)                                                                    Output Shape
=================================================================================================
VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))     (4, 4, 512)
_________________________________________________________________________________________________
Flatten()                                                                       (8192)
_________________________________________________________________________________________________
Dense(256, activation='relu')                                                   (256)
_________________________________________________________________________________________________
Dense(10, activation='softmax')                                                 (10)
=================================================================================================
```

Les deux phases d'entraînement afin d'implémenter le Fine-Tuning sont lancées comme suit :

```
conv_base.trainable = False

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(lr=3e-4),
              metrics=['acc'])
```

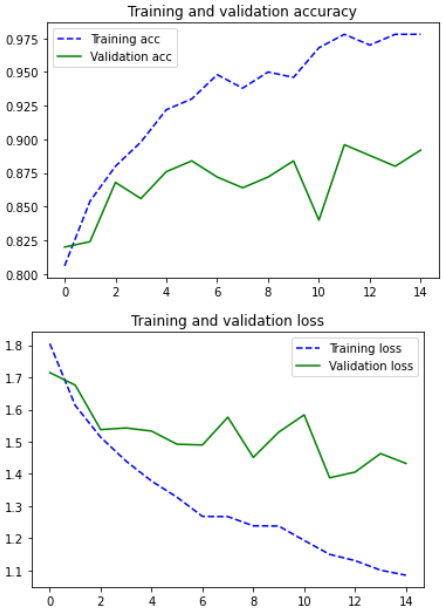
```
history1 = model.fit(train_generator_augmented,
                     validation_data=validation_generator,
                     epochs=15,
                     )


conv_base.trainable = True

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(lr=1e-5),
              metrics=['acc'])

history = model.fit(train_generator_augmented,
                     validation_data=validation_generator,
                     epochs=15,
                     )
```

Et nous donnent les résultats suivants ur la phase de Fine-Tuning :



Transfert-Learning et Fine-Tuning avec VGG16 sur les données augmentées

Nous arrivons enfin à atteindre 90% d'accuracy sur les données de validation, même si, malgré la régularisation L_2 qui a été ajoutée, le sur-apprentissage est encore un peu présent ; il faudrait tester des régularisation L_1 ou dropout pour voir leurs effets.

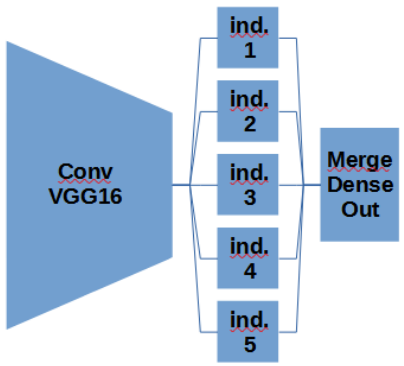
2.6 Test - Sagesse de la Foule

Nous avons en parallèle de tout cela voulu implémenter une architecture de réseau prenant parti de la théorie de la sagesse de la foule, qui présuppose que la perception et la résolution d'un problème

sont plus efficaces par une foule que par n'importe quel individu en faisant partie ou non.

2.6.1 Architectures

L'idée est d'avoir un réseau composé de multiples sous-réseaux jouant le rôle d'une foule, les résultats de ces multiples sous-réseaux sont ensuite donnés en entrée d'un dernier réseau qui les analyse et établit une sortie. En pratique, on récupère la partie convolutive de VGG16 pour encore tirer parti du Transfer-Learning, cette partie convolutive étant assez générale, elle est mise en commun, et c'est au niveau de la partie dense que se produit la division du réseau en différents individus ; chacun de ces "individus" est un sous-réseau dense qui prend en entrée la sortie de la base convolutive, les sorties de ces sous-réseaux est ensuite concaténée et donnée en entrée d'un dernier réseau, dense, qui renvoit la sortie du réseau global.



Architecture du réseau de foule.

Chaque sous réseau `ind.j` a pour architecture (un `Flatten()` étant fait à la sortie de la base convolutive) :

```
_________________________________________________
Layer (type)                 Output Shape
=================================================
Dense(50, activation='relu')        (50)
_________________________________________________
Dense(60, activation='relu')        (60)
_________________________________________________
Dense(10, activation='relu')     (10)
=================================================
```

Tandis que le réseau noté `Merge Dense Out` a pour architecture (un `concatenate(ind.j_list)` et `Flatten()` étant faits pour rassembler les sorties des sous réseaux) :

```
_________________________________________________
Layer (type)                 Output Shape
```

```
=================================================
Dense(512, activation='relu')       (512)
-------------------------------------------------
Dense(10, activation='softmax')     (10)
=================================================
```

Les réseaux ne sont soumis à aucune régularisation et nous avons fait nos tests pour `nb_agents = 5`.

2.6.2 Entraînement

La particularité des sous-réseaux est que nous souhaitons qu'ils aient des expériences différentes, pour ce faire chacun est entrainé (avec une activation `'softmax'` à la place de la `'relu'` indiquée dans l'architecture) sur une partie des données qui lui est propre. Pour ce faire l'algorithme de chargement des données a été modifié pour ajouter un sous dossier respectif à chaque sous réseau en plus du dossier `path/train/train` qui contient toutes les données ; dans un premier temps on charge les données de manière homogène dans chaque sous dossier (avec création des `dataframe` et `datagen` associés somme fait plus haut) :

```
j=0
for i in range(10):
  for filename in train_filenames[i]:
    img = Image.open(path_train+mushrooms_names[i]+"/"+filename).convert('RGB')
    img = img.resize((image_size, image_size))
    img.save(path_train+"train/"+filename)
    img.save(path_train+"train"+str(j%nb_agents)+"/"+filename)
    j+=1.trainable = False
```

Après plusieurs essais qu'il serait fastidieux de résumer ici, la marche à suivre pour l'entrainement est la suivante :

1. Entraînement de chaque sous réseau sur leur sous-dataset respectif :

```
for j in range(nb_agents):
    model_agents[j].compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(lr=3e-4),
              metrics=['acc'])

    history_agents.append(
      model_agents[j].fit(train_agents_generator_augmented[j],
                          validation_data=validation_generator,
                          epochs=10,
                          )

```

2. Entraînement de la dernière partie du réseau avec la base convolutive et chaque sous-réseau gelés :

```
conv_base.trainable = False
agents_pool.trainable = False

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(lr=3e-4),
              metrics=['acc'])

```

```
history = model.fit(train_generator_augmented,
                    validation_data=validation_generator,
                    epochs=5,
                    )

```

3. Fine-Tuning de la base convolutive :

```
conv_base.trainable = True

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(lr=1e-5),
              metrics=['acc'])

history = model.fit(train_generator_augmented,
                    validation_data=validation_generator,
                    epochs=7,
                    )

```

4. "Fine-Tuning" de l'ensemble du réseau :

```
agents_pool.trainable = True

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(lr=1e-5),
              metrics=['acc'])

history = model.fit(train_generator_augmented,
                    validation_data=validation_generator,
                    epochs=5,
                    )

```

Les résultats de ce réseau sur les données de test sont moins bonnes que le réseau basique de Transfert-Learning précédemment effectués : `loss: 65.62, acc: 82.80%`

2.6.3 Variété dans les sous-réseaux

Apporter de la variété dans les sous réseaux (en termes de données d'apprentissage) améliore légèrement ces résultats. Cette variété est ajoutée en sur-représentant certaines classes pour certains de ces sous-réseaux, comme suit (pour `nb_agents = 5`) :

```
j=0
for i in range(10):
  for filename in train_filenames[i]:
    img = Image.open(path_train+mushrooms_names[i]+"/"+filename).convert('RGB')
    img = img.resize((image_size, image_size))
    img.save(path_train+"train/"+filename)
    img.save(path_train+"train"+str(j%nb_agents)+"/"+filename)
    img.save(path_train+"train"+str(i//2)+"/"+filename)
    j+=1
```

Les résultats sont très légèrement améliorés : `loss: 49.52, acc: 84.40%`, ce qui tend à confirmer qu'une décision prises sur une base riche et variée sont plus abouties que celles prises sur une base uniforme.

3 Analyse des résultats

3.1 Analyse quantitative

3.1.1 Premier reseau standard

```
loss: 234.54%
acc: 58.80%
Confusion matrix
[[13  0  2  0  4  1  2  2  0  1]
 [ 0 15  0  1  0  4  0  1  3  1]
 [ 1  0 13  0  5  2  1  2  1  0]
 [ 0  0  2 16  0  4  0  1  0  2]
 [ 5  0  1  2 12  1  1  2  0  1]
 [ 2  0  5  1  1 10  1  3  0  2]
 [ 3  0  1  0  5  0 14  2  0  0]
 [ 4  1  0  0  0  0  3 15  0  2]
 [ 2  0  1  1  0  0  0  3 18  0]
 [ 1  0  1  1  0  1  0  0  0 21]]
```

Matrice de confusion du premier reseau

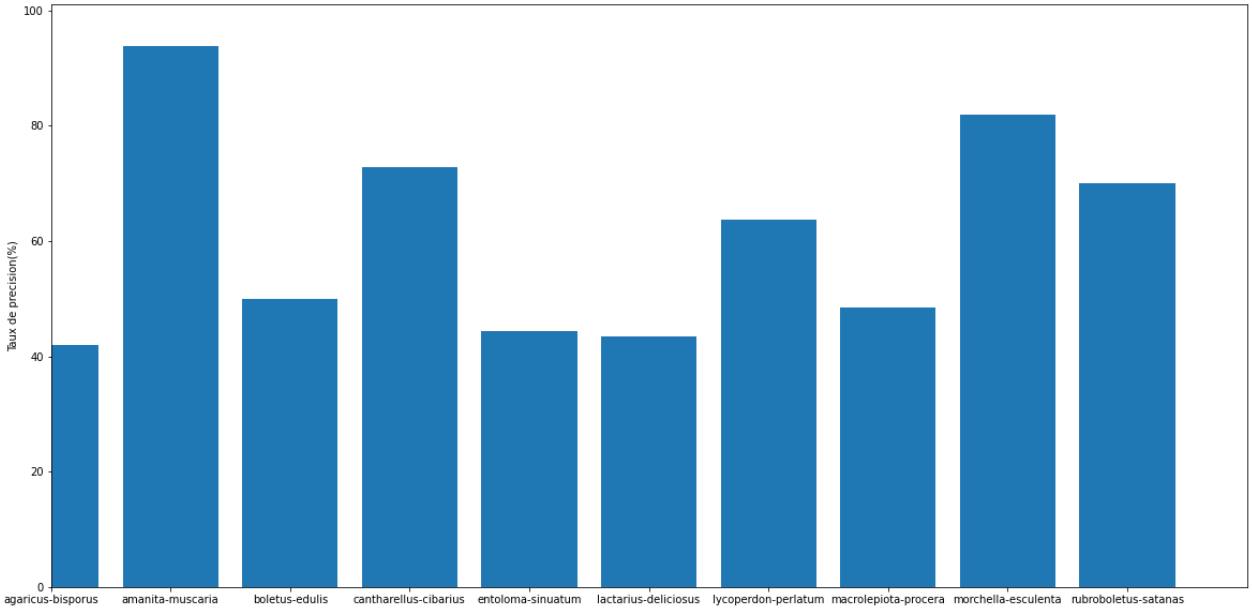


Diagramme de l'accuracy en fonction des classes

Analyse: L'accuracy globale est de 58.80% pour ce réseau. Ce qui n'est pas suffisamment performant, c'est pourquoi on effectuera de la régularisation pour rendre les resultats plus performants. Quant à l'accuracy par classe, on voit que selon les classes on a une nette différence d'accuracy, par exemple entre aminata-muscaria (95%) et agaricus-bisporus (42%). Cela est certainement due à la

base de données et à la présence ou non de caractéristiques évidents propores au champignon.

3.1.2 Second réseau: Augmentation des données

```
loss: 104.82%
acc: 70.80%
Confusion matrix
[[12  0  2  0  8  2  1  0  0  0]
 [ 0 20  3  1  0  0  0  0  0  1]
 [ 0  4 16  0  1  0  3  1  0  0]
 [ 0  0  0 18  0  6  0  1  0  0]
 [ 0  0  1  2 16  0  3  1  2  0]
 [ 1  2  5  1  2 14  0  0  0  0]
 [ 3  0  0  0  1  0 20  1  0  0]
 [ 1  0  3  0  0  1  1 17  2  0]
 [ 0  0  3  1  0  1  0  0 20  0]
 [ 0  1  0  0  0  0  0  0  0 24]]
```

Matrice de confusion du second reseau

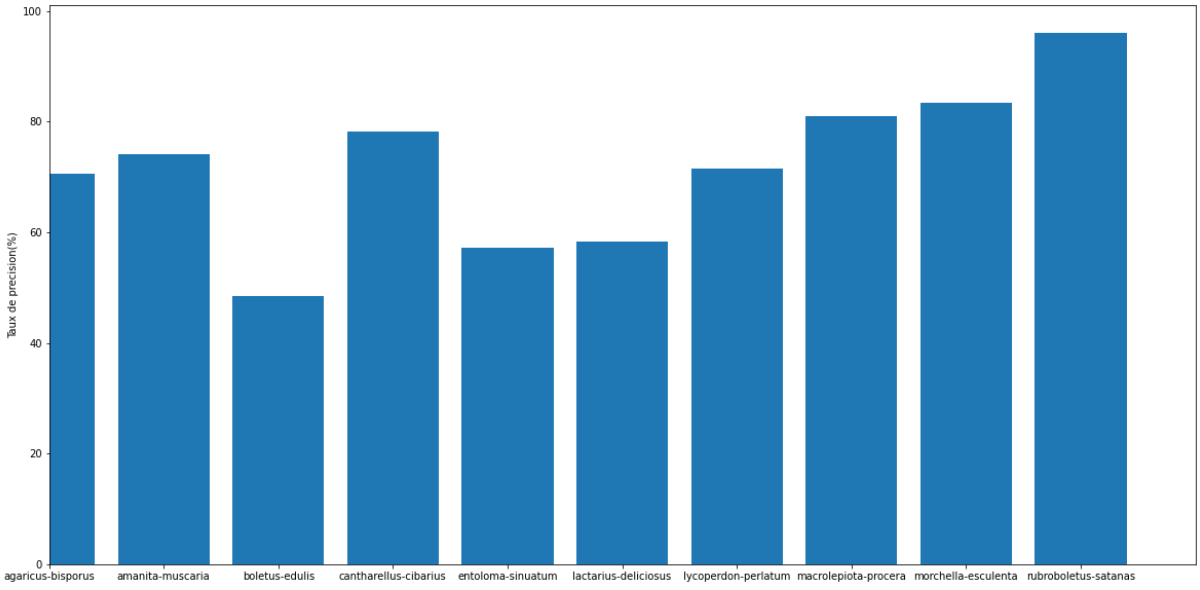


Diagramme de l'accuracy en fonction des classes

Analyse: L'accuracy globale est de 70.80% pour ce réseau. Ce qui n'est toujours pas suffisamment performant, cependant on voit que l'augmentation des données a permis d'améliorer la précision globale. Quant à l'accuracy par classe, on voit qu'elle varie toujours selon les classes, cependant cette variation a diminué par rapport au diagramme précédent. On remarque cependant

que le champignon boietus-edulis n'a pas bénéficié de l'augmentation de données, seulement 42% de précision.

3.1.3 Troisième réseau: Reseau Plus Complexe

```
loss: 111.89%
acc: 70.80%
Confusion matrix
[[ 8  0  2  0  8  2  0  4  1  0]
 [ 0 18  2  1  0  3  0  0  1  0]
 [ 1  1 16  1  3  0  0  1  0  2]
 [ 0  0  0 20  0  5  0  0  0  0]
 [ 3  0  1  1 17  0  0  1  2  0]
 [ 0  0  2  1  3 17  0  2  0  0]
 [ 0  0  0  0  8  1 14  2  0  0]
 [ 0  0  1  0  1  1  0 22  0  0]
 [ 0  0  0  0  0  2  0  0 23  0]
 [ 0  0  0  0  0  2  0  1  0 22]
```

Matrice de confusion du troisième réseau

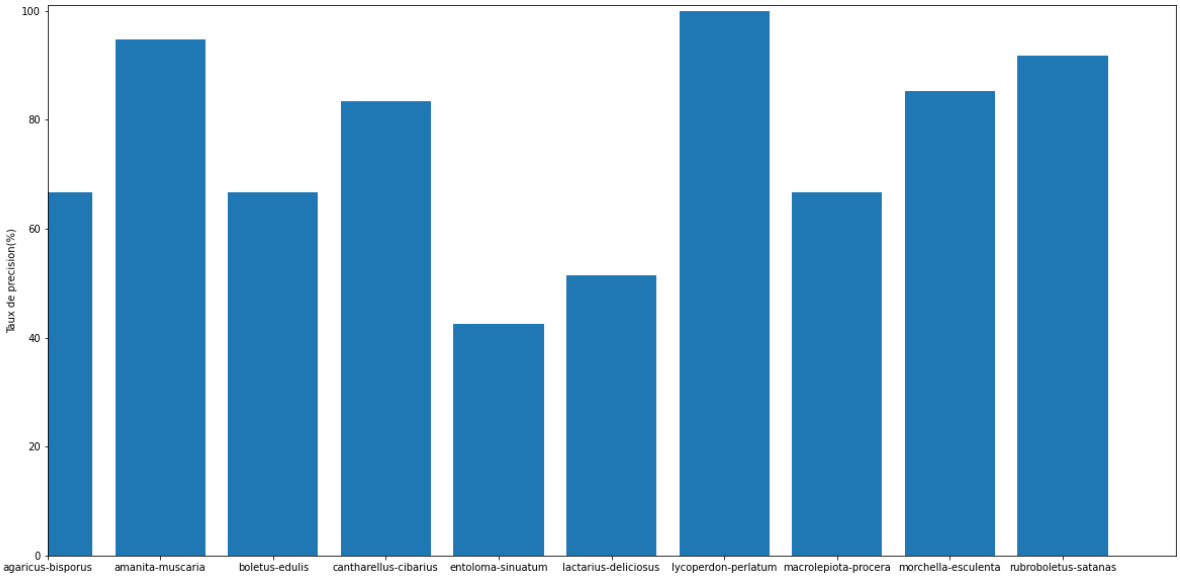


Diagramme de l'accuracy en fonction des classes

Analyse: L'accuracy globale est de 70.80% pour ce réseau. Ce qui est la même precision que l'ancien reseaux. Cependant Le reseau a de meilleur résultat par classe! Seul la classe entoloma-sinuatum est mal reconnu avec un taux de 40%.Enfaite comme expliqué dans la partie qualititative. Ce sont des champignons qui poussent en groupe et l'algorithme les classe dans agaricus-bisporus.On a donc progresser puisque le réseau reconnait mieux (a part pour celui la) les autres champignons qui sont seul.

3.1.4 Quatrième réseau: Transfert Learning-Fine Tuning

```
loss: 155.95%
acc: 85.60%
Confusion matrix
[[18  0  1  2  1  1  0  1  0  1]
 [ 0 24  0  0  0  1  0  0  0  0]
 [ 0  0 24  0  0  1  0  0  0  0]
 [ 0  0  0 21  0  4  0  0  0  0]
 [ 2  0  0  1 17  2  0  2  0  1]
 [ 2  0  1  0  1 19  0  2  0  0]
 [ 4  0  0  0  1  0 19  1  0  0]
 [ 0  1  1  0  0  0  0 23  0  0]
 [ 0  0  0  0  0  0  0  0 25  0]
 [ 0  0  1  0  0  0  0  0  0 24]]
```

Matrice de confusion du quatrième réseau

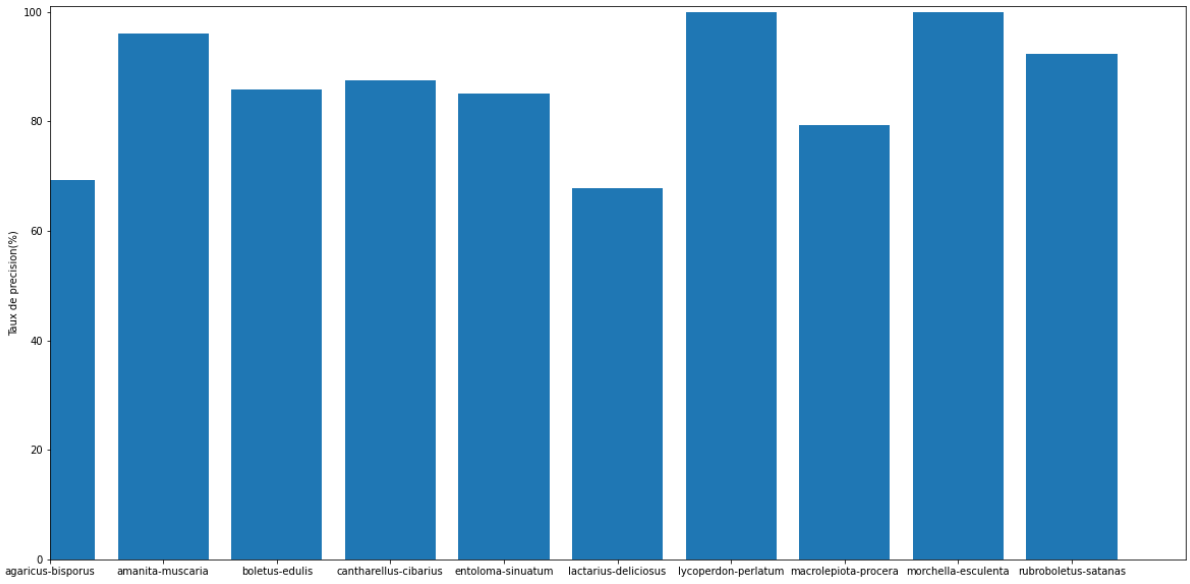


Diagramme de l'accuracy en fonction des classes

Analyse: L'accuracy globale est de 85.60% pour ce réseau. Ce qui est tout à fait correct. La méthode de Transfert Learning-Fine Tuning est très efficace. On voit même pour l'accuracy globale que la majorité des classes ont une précision dépassant 80%, certaines sont entre $90\% et 100\%$, ce qui est excellent. Il reconnait mieux les champignons qui poussent en groupe: agaricus-bisporus et entoloma-sinuatum. Le seul champignon posant problème, qui pourtant ne pousse pas en groupe est le dlactarius-deliciosus. On remarque dans la matrice de confusion, que ce champignon a tendance à être classé dans les 5 premières classes de facon dispersive, ce qui nous laisse penser que les images

de la base de données de ce champignon ne sont pas assez représentatives, car même dans les autres réseaux le taux de reconnaissance de ce dernier reste faible.

3.1.5 Cinquième réseau: Réseau en parallèle

```
loss: 65.62%
acc: 82.80%
Confusion matrix
[[17  0  1  1  3  1  0  1  0  1]
 [ 0 24  0  0  0  1  0  0  0  0]
 [ 0  0 22  0  1  2  0  0  0  0]
 [ 0  0  0 19  0  6  0  0  0  0]
 [ 0  0  2  0 18  3  1  0  0  1]
 [ 1  0  3  0  3 16  0  1  0  1]
 [ 1  0  0  0  1  0 21  2  0  0]
 [ 0  1  1  0  0  0  0 23  0  0]
 [ 0  1  0  0  0  0  0  0 24  0]
 [ 1  0  0  0  1  0  0  0  0 23]]
```

Matrice de confusion du cinquième réseau

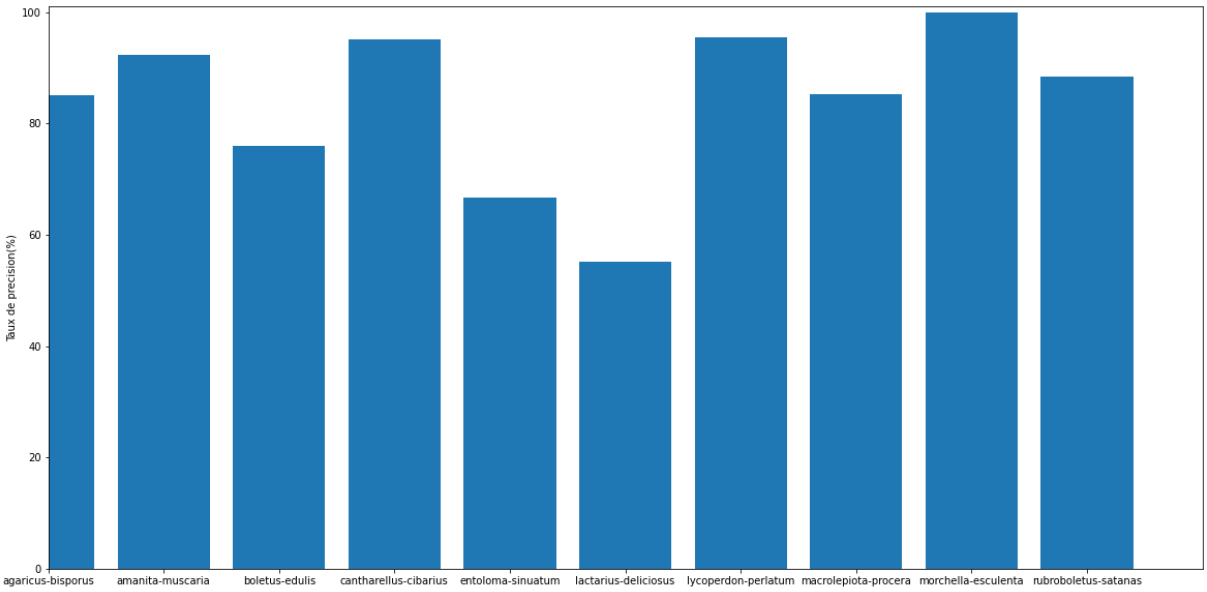


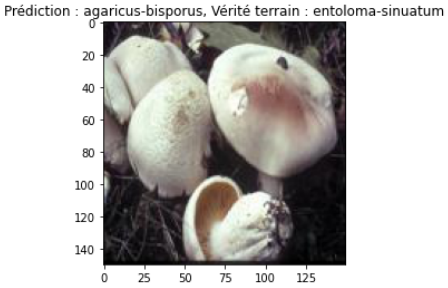
Diagramme de l'accuracy en fonction des classes

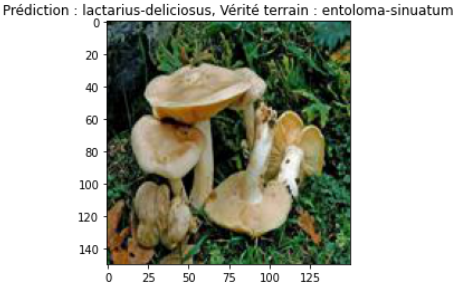
Analyse: L'accuracy globale est de 82.80% pour ce réseau. On remarque sur le diagramme de précision des classe que tout les taux sont meilleur qu'avant sauf le dlactarius-deliciosus.La encore on peut maitenant emmetre le de façon quasi certaine que cette partie de la base de donnée est a

refaire. Cette fois ci on constate une inversion de la tendance pour les champignons en groupe qui ont été plutot classé dans entoloma-sinuatum que dans agaricus-bisporus.Le reseau n'est manifestement dans ce cas d'une grande aide pour ameliorer la précision.

3.2 Analyse qualitative

Lorsque l'on observe les images qui ont été mal classifiées, on remarque que globalement beaucoup de ces erreurs sont des "cas limites". Voici deux exemples avec des photos des différentes variétés pour rappel :



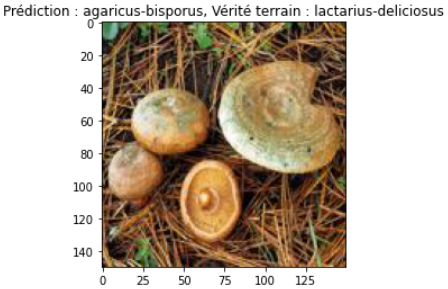


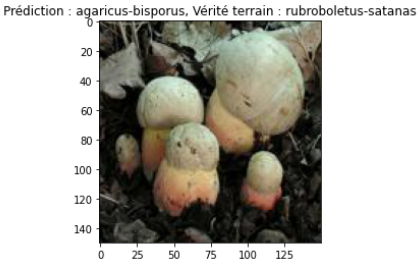
Agaricus bisporus

Entoloma sinuatum

Lactarius deliciosus

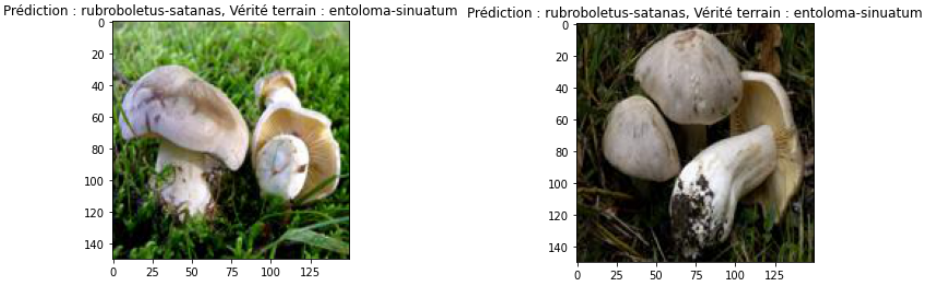
On observe également que les champignons sont souvent classifiés comme des Agaricus Bisporus lorsqu'ils sont présents en grand nombre sur l'image :



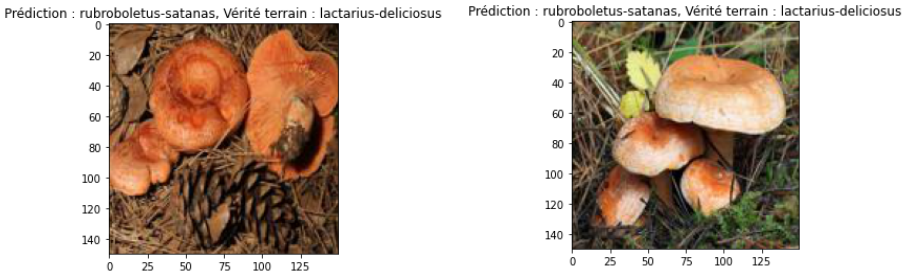


Ceci est dû au fait que les Agaricus Bisporus poussent par groupe et donc le réseau a appris à reconnaître cette caractéristique.

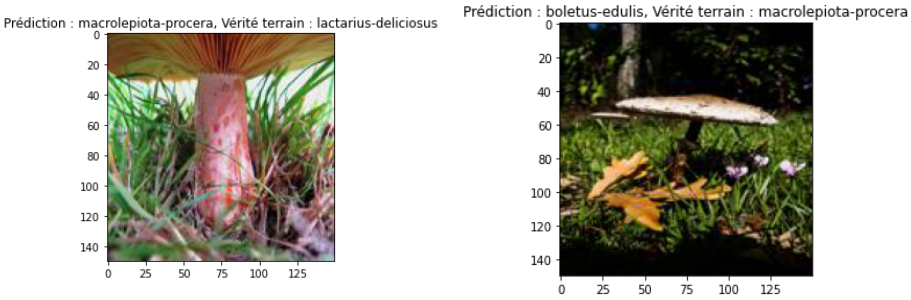
On observe le même comportement lorsque le réseau classifie des champignons ayant un gros pied comme étant des Rubroboletus Satanas :



Les Rubroboletus et les Lactarius étant tous les deux rouges, ils sont également parfois confondus :



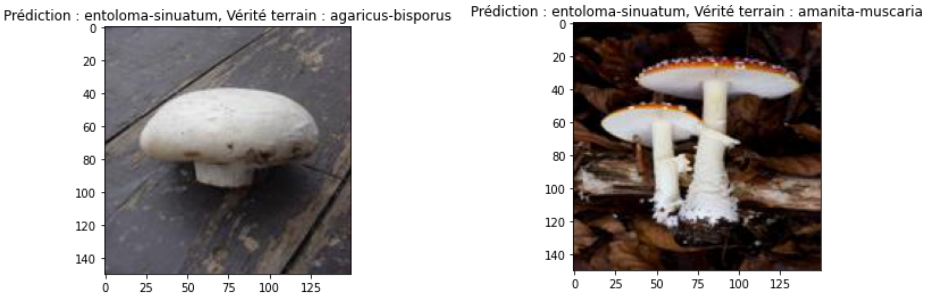
Enfin, quelques images de mauvaise qualité/ne permettant pas de bien voir le champignon sont également mal classifiées :



Certains de ces problèmes peuvent être corrigés en modifiant les images de la base de données, par exemple en supprimant les images de mauvaise qualité, en prenant des photos plus diversifiées de chaque champignon (plusieurs tailles de pied, plusieurs teintes...).

Le réseau fait rarement des erreurs sur des images clairement identifiables, bien que cela arrive

de temps en temps comme ici :



4 Conclusion

On remarque à l'aide des tests, que la base de données du champignon dlactarius-deliciosus n'est pas assez représentative de ce dernier car le réseau peine à le classer correctement. Il faudrait donc d'autres images pour ce champignon. On a remarqué des problèmes de précision, pour la classification de champignons poussant en groupe. Meme si nous avons réussi à palier ce problème dans le quatrième réseau, il serait peut être pertinent de ne prendre que des champignons seuls sur les images pour simplifier la reconnaissance. En enlevant un facteur de complications pour le réseau. En persepective on pourrait essayer d'ameliorer les premiers réseaux pour atteindre celui du transfer-Learning. Puisque le transfer-learning mis en place ici est un peu 'facile' au sens où on utilise un réseau déjà entrainé.