

Dans [...]

```

# IMPORTANT : EXÉCUTEZ CETTE CELLULE AFIN D'IMPORTER VOS SOURCES DE DONNÉES KAGGLE,
# PUIS N'HÉSITEZ PAS À SUPPRIMER CETTE CELLULE.
# REMARQUE : CET ENVIRONNEMENT DE PORTABLE DIFFÈRE DE L'ENVIRONNEMENT PYTHON DE KAGGLE
# IL PEUT DONC Y AVOIR DES BIBLIOTHÈQUES MANQUANTES UTILISÉES PAR VOTRE
# PORTABLE.
import kagglehub
import os

arnaudfadja_données_d'évaluation_de_la_qualité_et_des_défauts_des_prunes_africaines_path = kagglehub . dataset_download ( 'arnau
print ( 'Data source import complete.' )
print ( f "Le dataset a été téléchargé dans le dossier : { arnaudfadja_african_plums_quality_and_defect_assessment_data_path } " )

# Vous pouvez maintenant lister le contenu du dossier :
if arnaudfadja_african_plums_quality_and_defect_assessment_data_path :
    for item in os . listdir ( arnaudfadja_african_plums_quality_and_defect_assessment_data_path ) :
        print ( f " - { item } " )

    # Si vous savez qu'il y a un sous-dossier 'train' par exemple :
    train_folder = os . chemin . join ( arnaudfadja_african_plums_quality_and_defect_assessment_data_path , 'train' )
    if os . chemin . isdir ( train_folder ):
        print ( f "\n Contenu du dossier 'train' :" )
        pour l'élément dans os . listdir ( train_folder ) :
            print ( f "-- { item } " )

```

Importation de la source de données terminée.

Le jeu de données a été téléchargé dans le dossier : /kaggle/input/african-plums-quality-and-defect-assessment-data
- ensemble de données sur les prunes africaines

1) Importation des librairies et bibliothèques

Dans [...]

```

# Cet environnement Python 3 est livré avec de nombreuses bibliothèques d'analyse utiles installées
# Il est défini par l'image Docker kaggle/python : https://github.com/kaggle/docker-python
# Par exemple, voici plusieurs packages utiles à charger

importer numpy comme np # algèbre linéaire
importer pandas comme pd # traitement de données, fichier CSV E/S (par exemple pd.read_csv)
importer matplotlib.pyplot comme plt
depuis PIL importer Image
importer pandas comme pd

```

```

importer seaborn comme sns
importer random
importer shutil
depuis pathlib importer Path
depuis tqdm importer tqdm
importer torch
depuis torchvision importer datasets, transforms
depuis torchvision.datasets importer ImageFolder
depuis torch.utils.data importer Subset, Dataset, DataLoader
depuis torch importer nn, optim
depuis transformers importer AutoImageProcessor, ViTForImageClassification
depuis torch importer nn, optim
importer timm
depuis timm importer create_model
depuis threading importer Thread
depuis sklearn.metrics importer classification_report, confusion_matrix
importer torch.nn.functional comme F

# Les fichiers de données d'entrée sont disponibles dans le répertoire en lecture seule "../input/"
# Par exemple, l'exécution de cette commande (en cliquant sur Exécuter ou en appuyant sur Maj+Entrée) répertoriera tous les fichiers

#pour dirname, _, noms de fichiers dans os.walk('/kaggle/input/'):
# pour filename dans filenames:
# print(os.path.join(dirname, filename))

# Vous pouvez écrire jusqu'à 20 Go dans Le répertoire actuel (/kaggle/working/) qui est conservé comme sortie lorsque vous créez un fichier
# Vous pouvez également écrire des fichiers temporaires dans /kaggle/temp/, mais ils ne seront pas enregistrés en dehors de la session

```

Dans [...]

```
#éléments = os.listdir('/kaggle/working')
#imprimer(éléments)
```

2) Chargement des données

Dans [...]

```

#chemin racine du répertoire = arnaudfadja_african_plums_quality_and_defect_assessment_data_path + '/african_plums_dataset'

# Chemin du dossier contenant les sous-dossiers
dossier_images = os . chemin . rejoindre ( chemin , 'african_plums' )

# Obtenir les sous-dossiers

```

```

classes = [ os . chemin . join ( dossier_images , nom ) pour nom dans os . listdir ( dossier_images ) si os . chemin . is

# Créer une figure avec 1 ligne et 6 colonnes
fig , axes = plt . sous-intrigues ( 1 , 6 , figsize = ( 15 , 5 ))

# Afficher une image de chaque dossier
pour hache , dossier en zip ( axes , classes [: 6 ]): # Limiter à 6 dossiers
    # Charger la première image du dossier
    images = os . listdir ( dossier )
    si images :
        image_path = os . chemin . join ( dossier , images [ 55 ]) # Prendre la première image
        img = Image . ouvrez ( chemin_image )
        hache . imshow ( img )
        hache . axis ( 'off' ) # Masquer les axes
        axe . set_title ( os . path . basename ( dossier )) # Titre avec le nom du dossier

# Afficher la figure
plt . tight_layout ()
plt . show ()

```



Dans [...

```

# Chemin du dossier contenant le fichier CSV
nom_fichier = 'organized_plums_data_new.csv'

# Construire le chemin complet du fichier
chemin_fichier = os . chemin . join ( chemin , nom_fichier )

# Charger le fichier CSV dans un DataFrame
df = pd . read_csv ( chemin_fichier )

df . échantillon ( 10 )

```

Dehors[...]

	ID d'image	Étiquette	Type de défaut
4235	prune_non_mûre_555	vert	vert
2691	prune_non_affectée_732	bien	non affecté
2347	prune_non_affectée_388	bien	non affecté
503	prune_pourrie_23	défectueux	pourri
2109	prune_non_affectée_150	bien	non affecté
2248	prune_non_affectée_289	bien	non affecté
164	prune_brisée_165	défectueux	meurtri
3191	prune_non_affectée_1232	bien	non affecté
2787	prune_non_affectée_828	bien	non affecté
1197	prune_pourrie_717	défectueux	pourri

Dans [...]

```
df . forme
```

Dehors[...]

```
(4507, 3)
```

Dans [...]

```
df . columns = [ 'Image_Id' , 'Label' , 'class' ]  
df . sample ( 10 )
```

Dehors[...]

	Image_Id	Étiquette	classe
83	prune_brisée_84	défectueux	meurtri
3280	prune_non_affectée_1321	bien	non affecté
847	prune_pourrie_367	défectueux	pourri
3543	prune_non_affectée_1584	bien	non affecté
994	prune_pourrie_514	défectueux	pourri

	Image_Id	Étiquette	classe
3026	prune_non_affectée_1067	bien	non affecté
298	prune_brisée_299	défectueux	meurtri
3503	prune_non_affectée_1544	bien	non affecté
1811	prune_spottée_611	défectueux	repéré
3511	prune_non_affectée_1552	bien	non affecté

Dans [...

`df . info ()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex : 4 507 entrées, de 0 à 4 506
Colonnes de données (3 colonnes au total) :
 # Nombre de colonnes non nulles Type de données
-----
 0 Image_Id 4507 objet non nul
 1 Étiquette 4507 objet non nul
 2 objets non nuls de classe 4507
dtypes : objet(3)
utilisation de la mémoire : 105,8 Ko et plus
```

Dans [...

`df . isna () . somme (axe = 0)`

Dehors[...]

0**Image_Id** 0**Étiquette** 0**classe** 0**type de données** : int64

Dans [...

```
df . dupliqué () . sum ()
print ( f "Nombre total de valeurs dupliquées : { df . duplicated () . sum () } " )
```

Nombre total de valeurs dupliquées : 0

Dans [...

```
print ( "Nombre total d'image pour chaque type de classe :"
df . groupby ( 'classe' ) . taille ()
```

Nombre total d'image pour chaque type de classe :

Dehors[...]

0**classe****meurtri** 319**fissuré** 162**pourri** 720**repéré** 759**non affecté** 1721**vert** 826**type de données** : int64

Dans [...

```
df . groupby ( 'Label' ) [ 'class' ] . agg ( lambda x : list ( set ( x ))) . reset_index ()
```

Dehors[...]

Étiquette**classe****0** défectueux [pourri, tacheté, meurtri, fissuré]**1** bien [non affecté]**2** vert [vert]

Dans [...]

```
# Initialiser les listes pour les noms de dossiers et le nombre d'images
noms_dossiers = []
nombre_images = []

# Compter le nombre d'images dans chaque sous-dossier
pour nom in os.listdir(dossier_images):
    chemin_dossier = os.chemin.join(dossier_images, nom)
    si os.chemin.isdir(chemin_dossier):
        images = [f pour f in os.listdir(chemin_dossier) si f.inférieur().se termine par (('.png', '.jpg',
        noms_dossiers.append(nom)
        nombre_images.ajouter(len(images)))

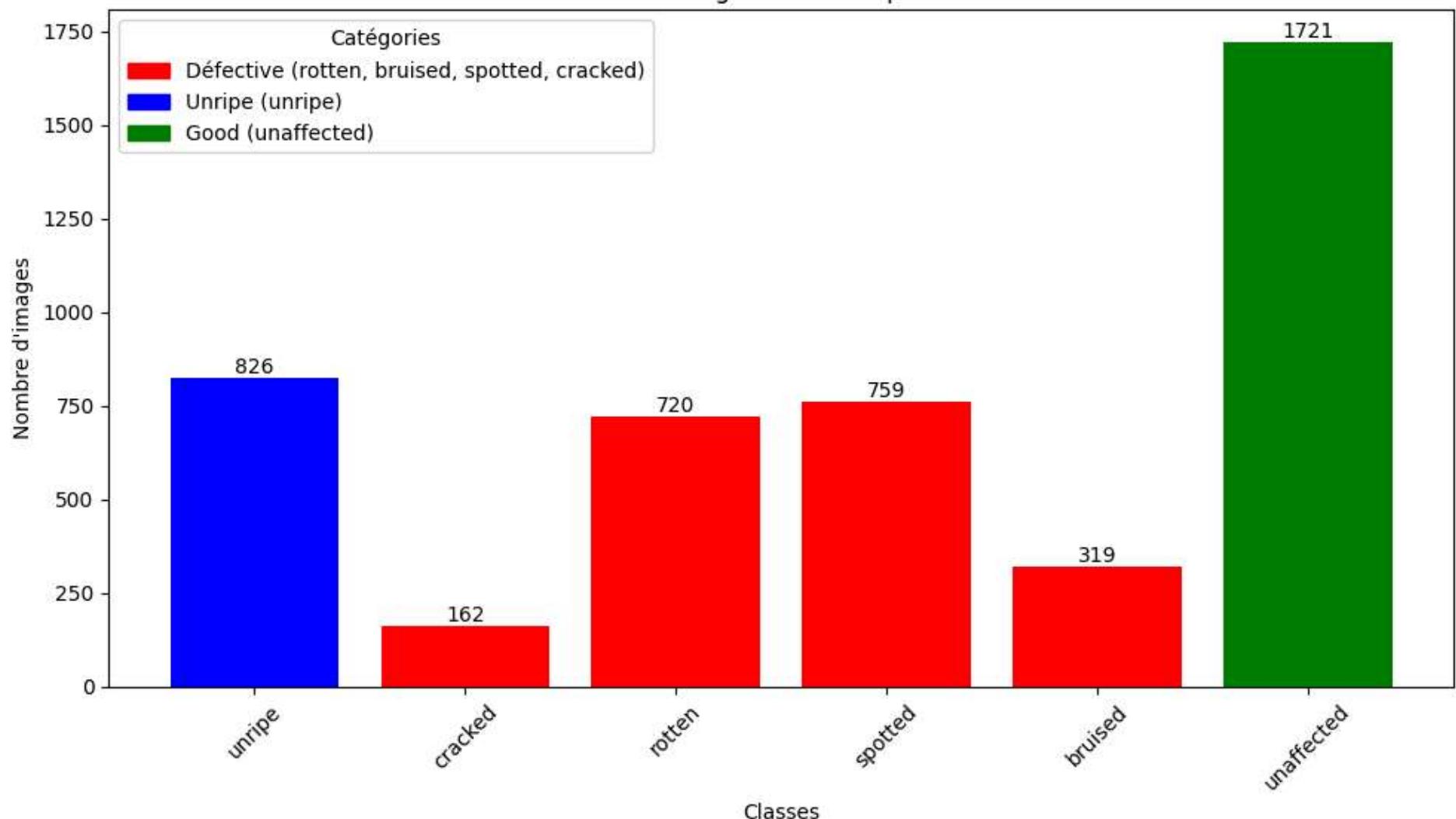
# Déterminer les couleurs des barres
couleurs = ['red' if nom in ['pourri', 'bleuisé', 'spotted', 'cracked'] else 'blue' if nom == 'immature' else 'green']

# Créer l'histogramme
plt.figure(figsize=(10, 6))
ax = plt.bar(noms_dossiers, nombre_images, color=couleurs)

# Ajouter des étiquettes sur les barres
plt.bar_label(ax)

# Configurer les axes et le titre
plt.xlabel('Classes')
plt.ylabel('Nombre d\'images')
plt.title('Nombre d\'images dans chaque dossier')
plt.xticks(rotation=45)
plt.serré_layout() # Ajuster l'espacement
# Ajouter la légende
plt.légende(
    poignées=[
        plt.Rectangle((0, 0), 1, 1, color='rouge', label='Défectueux (pourri, meurtri, tacheté, fissuré)'),
        plt.Rectangle((0, 0), 1, 1, color='bleu', label='Pas mûr (pas mûr)'),
        plt.Rectangle((0, 0), 1, 1, color='vert', label='Bon (non affecté)')
    ],
    titre='Catégories'
)
plt.show()
```

Nombre d'images dans chaque dossier



En résumé, les classes **tachetées, craquelées, pourries et meurtries** fondent toutes les parties de la classe **défectueuse**, et la classe **bonne** se confond avec la classe **non affectée**. Il est également à noter que les données sont très déséquilibrées mais nous allons gérer cela dans les étapes suivantes avec **l'augmentation des données**.

3) Prétraitement des images

Dans [...

```
# Petites fonctions pour connaître le nombre et les noms d'éléments dans un dossier
def compter_elements ( chemin , display_element = False ) :
    try :
        # Lister les éléments dans le répertoire
        elements = os . listdir ( chemin )
        # Compter le nombre d'éléments
        nombre_elements = len ( elements )

        if display_element :
            print ( f "Éléments du répertoire : { elements } " )

            print ( f "Nombre d'éléments : { nombre_elements } " )

    sauf FileNotFoundError :
        renvoie "Le chemin spécifié n'existe pas."
    sauf PermissionError :
        return "Accès refusé au chemin spécifié."
```

Dans [...

compter_elements (chemin , True)

```
Éléments du répertoire : ['README.md', 'organized_plums_data_new.csv', 'african_plums']
Nombre d'éléments : 3
```

Dans [...

chemin

Dehors[...]

'/kaggle/input/african-plums-quality-and-defect-assessment-data/african_plums_dataset'

Le principe général pour bien organiser nos données est le suivant :

1. Copier l'ensemble de données dans un dossier **data** (data_use) situer dans notre répertoire de travail (work_path) et y ajouter 2 dossiers **model_1** **et model_2** (folder_model_1 et dossier_model_2) qui contiennent les ensembles de données de chaque modèle.
2. A l'intérieur de chaque dossier de modèle, on craira d'abord 2 dossiers **train_val_temp_1(ou 2) et test_1(ou 2)** qui contiendront respectivement **90% et 10%** des données de chaque classe contenues dans le dossier data (tout en conservant leurs classes). Cad 90% de tacheté, 90% de mûr, ...
3. Consistant uniquement les dossiers test_1 et train_val_temp_1, on rassemble toutes les images contenues dans les dossiers **repérés, fissurés, pourris, et meurtris** dans un seul dossier du nom de **défectueux** afin d'avoir 3 classes.
4. Effectuer l'augmentation des données sur les données de test tout en équilibrant les différentes classes de sorte que les classes minoritaires prennent le même nombre d'image que la classe majoritaire.

5. Effectuer également l'augmentation des données sur les dossiers train_val_temp puis les diviser en dossiers **train et val** contenant respectivement **80% et 20% des 90% des images du dossier train_val_temp**.

3.1) Configuration des chemins

Dans [...

```
# Définir les chemins work_path = os.getcwd() path_data = os.path.join( path, 'african_plums' ) # Dossier source data_use
```

Dans [...

```
# Fonction pour créer un dossier (et le recréer s'il existe)
def create_folder( path ):
    if os . chemin . existe ( chemin ) :
        shul . rmmtree ( chemin )
        print ( f " 🗑️ Dossier existant supprimé : { chemin } " )
    os . makedirs ( chemin )
    print ( f " 📂 Nouveau dossier créé : { chemin } " )

# Création des dossiers nécessaires
print ( " 🔧 Création des dossiers requis..." )
create_folder ( data_use )
create_folder ( dossier_model_1 )
create_folder ( dossier_model_2 )
create_folder ( train_val_temp_1 )
create_folder ( train_val_temp_2 )
create_folder ( test_1 )
create_folder ( test_2 )

# Fonction pour copier les fichiers/dossiers avec barre de progression
def copy_with_progress ( src , dst ) :
    if not os . chemin . existe ( dst ):
        os . makedirs ( heure d'été )

    éléments = os . listdir ( src )
```

```
print ( f " \n 📁 Copie de { len ( items ) } éléments de ' { src } ' vers ' { dst } '... \n " )

pour l' élément dans tqdm ( éléments , desc = " 🚀 Copie en cours " , unit = " élément " )
: s = os.path.join ( src , élément ) d = os.path.join ( dst , élément ) si os.path.isdir ( s ) : shutil.copytree ( s , d
```

```
# Lancer la copie
copy_with_progress ( path_data , data_use )

print ( " \n ✅ Opération terminée avec succès ! Tous les fichiers ont été copiés 🎉 " )
```

📝 Création des dossiers requis...

- 📁 Nouveau dossier créé : /data
- 📁 Nouveau dossier créé : /model_1
- 📁 Nouveau dossier créé : /model_2
- 📁 Nouveau dossier créé : /model_1/train_val_temp_1
- 📁 Nouveau dossier créé : /model_2/train_val_temp_2
- 📁 Nouveau dossier créé : /model_1/test_1
- 📁 Nouveau dossier créé : /model_2/test_2

📁 Copie de 6 éléments de '/kaggle/input/african-plums-quality-and-defect-assessment-data/african_plums_dataset/african_plums' vers '/data'...

🚀 Copie en cours : 100% | ██████████ | 6/6 [00:29<00:00, 4.88s/élément]
✅ Opération terminée avec succès ! Tous les fichiers ont été copiés 🎉

Dans [...

```
compter_elements ( work_path , True )
compter_elements ( data_use , True )
compter_elements ( dossier_model_1 , True )
compter_elements ( dossier_model_2 , True )
```

Éléments du répertoire : ['.config', 'sample_data']

Nombre d'éléments : 2

Éléments du répertoire : ['taché', 'immature', 'non affecté', 'craqué', 'pourri', 'meurtri']

Nombre d'éléments : 6

Éléments du répertoire : ['test_1', 'train_val_temp_1']

Nombre d'éléments : 2

Éléments du répertoire : ['test_2', 'train_val_temp_2']
Nombre d'éléments : 2

3.2) Séparation des images dans les dossiers model_1 et model_2

Dans [...

```
# Classes à traiter
classes_model_1 = [ 'taché' , 'fissuré' , 'meurtri' , 'non affecté' , 'non mûr' , 'pourri' ]
classes_model_2 = [ 'taché' , 'fissuré' , 'meurtri' , 'pourri' ]

def split_images ( source_root , target_train , target_test , classes , ratio = 0.1 ) :
    pour class_name dans tqdm ( classes , desc = "🔍 Traitement des classes " , unit = " classe " ) :
        source_class_dir = os.path.join ( source_root , class_name ) images = os.listdir ( source_class_dir ) random.shuffle ( i

        test_count = int ( len ( images ) * ratio )
        test_images = images [ : test_count ]
        train_images = images [ test_count :]

        # Dossiers cibles
        class_train_dir = os . path . join ( target_train , class_name )
        class_test_dir = os . path . join ( target_test , class_name )

        os.makedirs ( class_train_dir , exist_ok = True ) os.makedirs ( class_test_dir , exist_ok = True )

        # Copier les images de test
        pour img dans test_images :
            src = os . path . join ( source_class_dir , img )
            dst = os . path . join ( class_test_dir , img )
            shutil . copy2 ( src , dst )

        # Copier les images de train_val
        pour img dans train_images :
            src = os . path . join ( source_class_dir , img )
            dst = os . path . join ( class_train_dir , img )
            shutil . copy2 ( src , dst )

        print ( f "📁 Classe '{ class_name }' : { len ( train_images ) } entraînement/val | { len ( test_images ) } test" )

    # Séparation pour le model_1
    print ( " \n 🎉 Séparation pour le model_1" )
```

```
split_images ( data_use , train_val_temp_1 , test_1 , classes_model_1 )

# Séparation pour le model_2
print ( " \n 🚀 Séparation pour le model_2" )
split_images ( data_use , train_val_temp_2 , test_2 , classes_model_2 )

print ( " \n ✅ Séparation terminée avec succès pour les deux modèles ! 🎉" )
```

🚀 Séparation pour le model_1

```
🔍 Traitement des classes : 17%|██████ | 1/6 [00:00<00:00, 9.86classe/s]
📁 Classe 'repéré' : 684 entraînement/val | 75 épreuves
📁 Classe 'craquée' : 146 entraînement/val | 16 épreuves
📁 Classe 'bleutée' : 288 entraînement/val | 31 épreuves
🔍 Traitement des classes : 83%|██████████ | 5/6 [00:00<00:00, 6.13cours/s]
📁 Classe 'non concerné' : 1549 entraînement/val | 172 épreuves
📁 Classe 'immature' : 744 entraînement/val | 82 essais
🔍 Traitement des classes : 100%|██████████ | 6/6 [00:00<00:00, 6.26cours/s]
📁 Classe 'pourri' : 648 entraînement/val | 72 essais
```

🚀 Séparation pour le model_2

```
🔍 Traitement des classes : 25%|██ | 1/4 [00:00<00:00, 6.80classe/s]
📁 Classe 'repéré' : 684 entraînement/val | 75 épreuves
📁 Classe 'craquée' : 146 entraînement/val | 16 épreuves
📁 Classe 'bleutée' : 288 entraînement/val | 31 épreuves
🔍 Traitement des classes : 100%|██████████ | 4/4 [00:00<00:00, 10.27cours/s]
📁 Classe 'pourri' : 648 entraînement/val | 72 essais
```

✅ Séparation terminée avec succès pour les deux modèles ! 🎉

Dans [...]

```
compter_elements ( train_val_temp_1 , True )
compter_elements ( test_1 , True )
compter_elements ( train_val_temp_2 , True )
compter_elements ( test_2 , True )
```

Éléments du répertoire : ['taché', 'immature', 'non affecté', 'craqué', 'pourri', 'meurtri']

Nombre d'éléments : 6

Éléments du répertoire : ['taché', 'immature', 'non affecté', 'craqué', 'pourri', 'meurtri']

Nombre d'éléments : 6

Éléments du répertoire : ['taché', 'craqué', 'pourri', 'meurtri']

Nombre d'éléments : 4

Éléments du répertoire : ['taché', 'craqué', 'pourri', 'meurtri']
 Nombre d'éléments : 4

3.3) Création des classes 'défectueuses'

Dans [...

```
# Dossiers à fusionner
défectueux_classes = [ 'spotted' , 'cracked' , 'bruised' , 'rotten' ]

# Chemins vers les sous-dossiers de model_1
#train_val_temp_1 = os.path.join('model_1', 'train_val_temp_1')
#test_1 = os.path.join('model_1', 'test_1')

def merge_to_defective ( root_dir )
    faulty_path = os.path.join ( root_dir , ' defective ' ) os.makedirs ( faultful_path , exist_ok = True )

    # Liste dynamique des classes à fusionner à partir du contenu réel
    class_names = os . listdir ( root_dir )
    pour class_name dans class_names :
        class_path = os . chemin . rejoindre ( rép_racine , nom_classe )

        # On ne fusionne que les classes "défectueuses"
        if class_name in [ 'spotted' , 'cracked' , 'bruised' , 'rotten' ] and os . path . isdir ( class_path ):
            images = os . listdir ( class_path )
            for img in tqdm ( images , desc = f "\ud83d\udcbb Fusion { class_name } \rightarrow 'defective' ( { root_dir } )" , unit = "img" )
                src = os . path . join ( class_path , img )
                dst = os . path . join ( faultueuse_path , img )

                # Renommage si doublon
                if os . path . exists ( dst ):
                    base , ext = os . path . splitext ( img )
                    count = 1
                    while os . path . exists ( dst ):
                        dst = os . path . join ( faulty_path , f " { base } _ { count } { ext } " )
                        count += 1

                    shutil . move ( src , dst )

                # Supprimer le dossier une fois fusion terminée
                fermé . rmtree ( class_path )
                print ( f "\ud83d\udcbb Supprimé : { class_path } " )
```

```

        elif class_name in [ 'unaffected' , 'unripe' ]:
            print ( f " ✅ Conservé : { class_name } " )

        print ( f " \n ✅ Structure finale dans { root_dir } : { os . listdir ( root_dir ) } " )

# Appliquer pour train_val_temp_1 et test_1
print ( " \n 🚀 Fusion des classes défectueuses dans 'train_val_temp_1'" )
merge_to_defective ( train_val_temp_1 )

print ( " \n 🚀 Fusion des classes défectueuses dans 'test_1'" )
merge_to_defective ( test_1 )

print ( " \n ✅ Fusion terminée avec succès pour model_1 ! 🎉 " )

```

🚀 Fusion des classes défectueuses dans 'train_val_temp_1'

Fusion 'repéré' → 'défectueux' (/model_1/train_val_temp_1) : 100 %|██████████| 684/684 [00:00<00:00, 31 490,77 img/s]

Supprimer : /model_1/train_val_temp_1/spotted

Conservé : non mûr

Conservé : non affecté

Fusion 'fissuré' → 'défectueux' (/model_1/train_val_temp_1) : 100 %|██████████| 146/146 [00:00<00:00, 28 122,54 img/s]

Supprimer : /model_1/train_val_temp_1/cracked

Fusion 'pourri' → 'défectueux' (/model_1/train_val_temp_1) : 100 %|██████████| 648/648 [00:00<00:00, 35 897,52 img/s]

Supprimé : /model_1/train_val_temp_1/rotten

Fusion 'meurtri' → 'défectueux' (/model_1/train_val_temp_1) : 100 %|██████████| 288/288 [00:00<00:00, 31 952,38 img/s]

Supprimé : /model_1/train_val_temp_1/bruised

Structure finale dans /model_1/train_val_temp_1 : ['unripe', 'unaffected', 'defective']

🚀 Fusion des classes défectueuses dans 'test_1'

Fusion 'repéré' → 'défectueux' (/model_1/test_1) : 100 %|██████████| 75/75 [00:00<00:00, 25 206,15 img/s]

Supprimé : /model_1/test_1/spotted

Conservé : non mûr

Conservé : non affecté

Fusion 'fissuré' → 'défectueux' (/model_1/test_1) : 100 %|██████████| 16/16 [00:00<00:00, 19256.49img/s]

Supprimé : /model_1/test_1/cracked

Fusion 'pourri' → 'défectueux' (/model_1/test_1) : 100 %|██████████| 72/72 [00:00<00:00, 22874.56img/s]

Supprimé : /model_1/test_1/rotten

Fusion « meurtri » → « défectueux » (/model_1/test_1) : 100 %|██████████| 31/31 [00:00<00:00, 20873.88img/s]

Supprimé : /model_1/test_1/bruised

Structure finale dans /model_1/test_1 : ['immature', 'unaffected', 'defective']

✓ Fusion terminée avec succès pour model_1 ! 🎉

Dans [...

```
compter_elements ( train_val_temp_1 ,  True )
compter_elements ( test_1 ,  True )
compter_elements ( train_val_temp_2 ,  True )
compter_elements ( test_2 ,  True )
```

Éléments du répertoire : ['immature', 'non affecté', 'défectueux']

Nombre d'éléments : 3

Éléments du répertoire : ['immature', 'non affecté', 'défectueux']

Nombre d'éléments : 3

Éléments du répertoire : ['taché', 'craqué', 'pourri', 'meurtri']

Nombre d'éléments : 4

Éléments du répertoire : ['taché', 'craqué', 'pourri', 'meurtri']

Nombre d'éléments : 4

Dans [...

```
compter_elements ( train_val_temp_1 ,  False )
compter_elements ( train_val_temp_2 ,  False )
```

Nombre d'éléments : 3

Nombre d'éléments : 4

3.4) Data augmentation, équilibrage des données et création des dossiers train, val des différents dossiers model_1 et model_2

Dans [...

```
def  get_transforms_1 (
    ) : return transforms.Compose ( [ transforms.RandomHorizontalFlip ( ) , transforms.RandomVerticalFlip ( ) , transforms.Random
```

Dans [...

```
def  get_transforms_2 () :
    return transforms.Compose ( [ transforms.RandomHorizontalFlip ( ) , transforms.RandomVerticalFlip ( ) , transforms.RandomRota
```

Augmentation et équilibrage des dossiers du modèle 2

```
Dans [ ... ]:
```

```
def balance_dataset_with_augmentation ( dataset_root ) :
    transform = get_transforms_2 ()

    nombre_de_classes = {}
    images_de_classes = {}

    pour cls dans os . listdir ( dataset_path ) :
        cls_path = os . path . join ( dataset_root , cls )
        sinon os . path . isdir ( cls_path ) : continue
        images = [ img pour img dans os . listdir ( cls_path ) si img . lower () .
```



```
if not class_counts :
    print ( f "⚠️ Aucune image trouvée dans { dataset_root } ." )
    return

max_count = max ( class_counts.values ( ) )

# Règle dynamique pour le target_count
if 'train_val_temp' in dataset_root et max_count < 1500 :
    target_count = 1500
elif 'test' in dataset_root et max_count < 150 :
    target_count = 150
else :
    target_count = max ( max_count , 150 ) # Par défaut on garde la logique précédente

print ( f " \n 📊 Nombre d'images cibles par classe : { target_count } (dans { dataset_root } )" )

pour cls dans tqdm ( class_images , desc = "🔄 Équilibrage en cours" , unit = "classe" ):
```

```

cls_path = os . path . join ( dataset_root , cls )
images = class_images [ cls ]
current_count = len ( images )
i = 0

# Ne pas augmenter si déjà suffisant et
pas dans un cas spécial if current_count >= target_count and 'train_val_temp' not in dataset_root : print ( f "✅ C]

to_generate = target_count - current_count
print ( f "⚠ Classe '{ cls }' → { current_count } → { target_count } (À propos de { to_generate } images)" )

tant que current_count < target_count :
    img_name = random . choice ( images )
    img_path = os . path . join ( cls_path , img_name )

    essayer :
        avec Image . open ( img_path ) . convert ( "RGB" ) comme image :
            aug_img = transform ( image )
            aug_filename = f "{ os . path . splitext ( img_name )[ 0 ] } _aug { i } .png"
            aug_img . save ( os . path . join ( cls_path , aug_filename ))
            current_count += 1
            i += 1
        sauf Exception comme e :
            print ( f "⚠ Erreur sur { img_path } : { e } " )
            continuer

    print ( f "\n ✅ Équilibrage terminé pour : { dataset_root } \n " )

```

Dans [...

```

balance_dataset_with_augmentation ( train_val_temp_2 )
balance_dataset_with_augmentation ( test_2 )

```

📊 Nombre d'images cibles par classe : 1500 (dans /model_2/train_val_temp_2)

⌚ Équilibrage en cours : 0% | 0/4 [00:00<?, ?classe/s]

⚠ Classe 'repéré' → 684 → 1500 (Ajout de 816 images)

⌚ Équilibrage en cours : 25% | 1/4 [01:02<03:07, 62,39s/cours]

⚠ Classe 'craquée' → 146 → 1500 (Ajout de 1354 images)

⌚ Équilibrage en cours : 50% | 2/4 [02:55<03:03, 91,97s/cours]

⚠ Classe 'pourri' → 648 → 1500 (Ajout de 852 images)

⌚ Équilibrage en cours : 75% | 3/4 [03:54<01:17, 77,18s/cours]

↻ Classe 'meurtri' → 288 → 1500 (Ajout de 1212 images)
⟳ Équilibrage en cours : 100% |██████████| 4/4 [05:32<00:00, 83.04s/cours]
✓ Équilibrage terminé pour : /model 2/train val temp 2

```
[!] Nombre d'images cibles par classe : 150 (dans /model_2/test_2)
[?] Équilibrage en cours : 0% | 0/4 [00:00<?, ?classe/s]
[?] Classe 'repéré' → 75 → 150 (Ajout de 75 images)
[?] Équilibrage en cours : 25% | [■] | 1/4 [00:06<00:19, 6,65s/cours]
[?] Classe 'craquée' → 16 → 150 (Ajout de 134 images)
[?] Équilibrage en cours : 50% | [■] | 2/4 [00:15<00:15, 7.98s/cours]
[?] Classe 'pourri' → 72 → 150 (Ajout de 78 images)
[?] Équilibrage en cours : 75% | [■] | 3/4 [00:21<00:06, 6,86s/cours]
[?] Classe 'bleutée' → 31 → 150 (Ajout de 119 images)
[?] Équilibrage en cours : 100% | [■] | 4/4 [00:28<00:00, 7.12s]
[✓] Équilibrage terminé pour : /model_2/test_2
```

Augmentation et équilibrage des données de train_val_temp pour création de train et val

Dans [...] ...

```
def augment_to_balance ( input_dir ):
    transform = get_transforms_1 ()

    nombre_de_classes = {}
    images_de_classes = {}

    pour cls dans os . listdir ( input_dir ):
        cls_path = os . path . join ( input_dir , cls )
        sinon os . path . isdir ( cls_path ): continue images = [ img pour img dans os . listdir ( cls_path ) si img . lower () == classe . lower () : images . append ( img ) ; class_counts [ classe ] += 1 ]
    fin

    if not class_counts :
        print ( f "⚠ Aucune image trouvée dans { input_dir } . Traitement ignoré." )
        return

    max_count = max ( class_counts .values () ) target_count = max ( max_count , 150 ) print ( f " \n 📈 Classe la plus représentante : { classe } avec { max_count } images . Cible : { target_count } images . " )
    for classe in class_counts .keys () :
        if class_counts [ classe ] < target_count :
            for i in range ( target_count - class_counts [ classe ] ) :
                img = choice ( images [ classe ] )
                transform . __call__ ( img )
                img . save ( os . path . join ( output_dir , classe , img . name ) )

```

```

pour cls dans tqdm ( class_images , desc = f "⚠️ Augmentation dans { input_dir } " , unit = "classe" ):
    cls_path = os . path . join ( input_dir , cls )
    images = class_images [ cls ]
    current_count = len ( images )
    i = 0

    si current_count >= target_count et 'train_val_temp' n'est pas dans input_dir :
        print ( f "✅ Classe '{ cls } ' déjà équilibrée ( { current_count } images)" )
        continue

    while current_count < target_count :
        img_name = random . choice ( images )
        img_path = os . path . join ( cls_path , img_name )
        try :
            image = Image . open ( img_path ) . convert ( "RGB" )
            aug_img = transform ( image )
            aug_name = f "{ os . path . splitext ( img_name )[ 0 ] } _aug { i } .png"
            aug_path = os . path . join ( cls_path , aug_name )
            aug_img . save ( aug_path )
            current_count += 1
            i += 1
        except Exception as e :
            print ( f "⚠️ Erreur sur { img_path } : { e } " )
            continue

    print(f"✅ Data augmentation terminée pour {input_dir} 🎉")

```

📁 Exécution sur les deux dossiers :

```

augment_to_balance(train_val_temp_1)
augment_to_balance(test_1)

```

📊 Classe la plus représentée initialement dans /model_1/train_val_temp_1 : 1766 images
🎯 Nombre cible d'images par classe dans /model_1/train_val_temp_1 : 1766 images

⚠️ Augmentation dans /model_1/train_val_temp_1: 100%|██████████| 3/3 [00:57<00:00, 19.25s/classe]
✅ Data augmentation terminée pour /model_1/train_val_temp_1 🎉

📊 Classe la plus représentée initialement dans /model_1/test_1 : 194 images
🎯 Nombre cible d'images par classe dans /model_1/test_1 : 194 images

⚠️ Augmentation dans /model_1/test_1: 100%|██████████| 3/3 [00:08<00:00, 2.80s/classe]

- Classe 'defective' déjà équilibrée (194 images)
- Data augmentation terminée pour /model_1/test_1 🎉

In []:

```
def split_train_val(input_dir, output_dir, train_ratio=0.8):
    """
        Sépare les images de chaque classe dans un dossier en 80% train / 20% val.
        :param input_dir: train_val_temp_1 ou train_val_temp_2
        :param output_dir: model_1 ou model_2
        :param train_ratio: proportion des données pour l'entraînement
    """
    # Création des dossiers "train" et "val"
    train_dir = os.path.join(output_dir, 'train')
    val_dir = os.path.join(output_dir, 'val')
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir, exist_ok=True)

    # Parcours des classes présentes dans le dossier temporaire
    for class_name in os.listdir(input_dir):
        class_path = os.path.join(input_dir, class_name)
        if not os.path.isdir(class_path):
            continue

        # Liste des images
        images = [img for img in os.listdir(class_path)
                  if img.lower().endswith('.jpg', '.jpeg', '.png')]
        random.shuffle(images)

        # Split des images
        split_index = int(len(images) * train_ratio)
        train_images = images[:split_index]
        val_images = images[split_index:]

        # Création des dossiers de sortie pour chaque classe
        train_class_dir = os.path.join(train_dir, class_name)
        val_class_dir = os.path.join(val_dir, class_name)
        os.makedirs(train_class_dir, exist_ok=True)
        os.makedirs(val_class_dir, exist_ok=True)

        # Déplacement des images vers le dossier train
        for img in tqdm(train_images, desc=f"📁 {class_name} → train", unit="img"):
            shutil.move(os.path.join(class_path, img), os.path.join(train_class_dir, img))

        # Déplacement des images vers le dossier val
        for img in tqdm(val_images, desc=f"📁 {class_name} → val", unit="img"):
            shutil.move(os.path.join(class_path, img), os.path.join(val_class_dir, img))
```

```

for img in tqdm ( val_images , desc = f "📁 { class_name } → val" , unit = "img" ):
    shutil . move ( os . path . join ( class_path , img ) , os . path . join ( val_class_dir , img ) )

# Nettoyage du dossier temporaire une fois le split terminé
print ( f " \n 🗑 Suppression du dossier temporaire : { input_dir } " )
shutdown . rmtree ( input_dir )
print ( f " ✅ Split terminé pour { output_dir } ! 🎉 " )

# 🌟 Application aux deux modèles
split_train_val ( train_val_temp_1 , dossier_model_1 )
split_train_val ( train_val_temp_2 , dossier_model_2 )

```

📁 pas mûr → train : 100 % | ██████████ | 1412/1412 [00:00<00:00, 40806.98img/s]
 📁 pas mûr → val : 100 % ██████████ | 354/354 [00:00<00:00, 41113,80 img/s]
 📁 non affecté → train : 100 %|██████████| 1412/1412 [00:00<00:00, 39740.16img/s]
 📁 non affecté → val : 100 %|██████████| 354/354 [00:00<00:00, 28 615,72 img/s]
 📁 défectueux → train : 100 %|██████████| 1412/1412 [00:00<00:00, 37880.54img/s]
 📁 défectueux → val : 100 %|██████████| 354/354 [00:00<00:00, 36 736,61 img/s]

🗂 Suppression du dossier temporaire : /model_1/train_val_temp_1

✅ Split terminé pour /model_1 ! 🎉

📁 repéré → train : 100 % | ██████████ | 1200/1200 [00:00<00:00, 38127.73img/s]
 📁 repéré → val : 100 %|██████████| 300/300 [00:00<00:00, 32 987,92 img/s]
 📁 fissuré → train : 100 %|██████████| 1200/1200 [00:00<00:00, 39377.28img/s]
 📁 fissuré → val : 100 %|██████████| 300/300 [00:00<00:00, 32 784,22 img/s]
 📁 pourri → train : 100 %|██████████| 1200/1200 [00:00<00:00, 36093.75img/s]
 📁 pourri → val : 100 %|██████████| 300/300 [00:00<00:00, 27 398,83 img/s]
 📁 meurtri → train : 100 %|██████████| 1200/1200 [00:00<00:00, 36802.36img/s]
 📁 meurtri → val : 100 %|██████████| 300/300 [00:00<00:00, 36 662,43 img/s]

🗂 Suppression du dossier temporaire : /model_2/train_val_temp_2

✅ Split terminé pour /model_2 ! 🎉

Dans [...

```
compter_elements ( train_val_temp_1 , False )
```

Dehors[...]

"Le chemin spécifié n'existe pas."

Dans [...

```
compter_elements ( train_val_temp_2 , False )
```

Dehors[...]

"Le chemin spécifié n'existe pas."

Dans [...

```
compter_elements ( dossier_model_1 , True )
compter_elements ( dossier_model_1 + '/train' , True )
compter_elements ( dossier_model_1 + '/val' , True )
compter_elements ( dossier_model_1 + '/test_1' , True )
compter_elements ( dossier_model_2 , True )
compter_elements ( dossier_model_2 + '/train' , True )
compter_elements ( dossier_model_2 + '/val' , True )
compter_elements ( dossier_model_2 + '/test_2' , True )
```

Éléments du répertoire : ['test_1', 'train', 'val']

Nombre d'éléments : 3

Éléments du répertoire : ['immature', 'non affecté', 'défectueux']

Nombre d'éléments : 3

Éléments du répertoire : ['immature', 'non affecté', 'défectueux']

Nombre d'éléments : 3

Éléments du répertoire : ['immature', 'non affecté', 'défectueux']

Nombre d'éléments : 3

Éléments du répertoire : ['test_2', 'train', 'val']

Nombre d'éléments : 3

Éléments du répertoire : ['taché', 'craqué', 'pourri', 'meurtri']

Nombre d'éléments : 4

Éléments du répertoire : ['taché', 'craqué', 'pourri', 'meurtri']

Nombre d'éléments : 4

Éléments du répertoire : ['taché', 'craqué', 'pourri', 'meurtri']

Nombre d'éléments : 4

Dans [...

Vérifications si les ensembles de données sont équilibrés

```
cls_1 = [ 'non affecté' , 'défectueux' , 'pas mûr' ]
cls_2 = [ 'taché' , 'fissuré' , 'meurtri' , 'pourri' ]
pour i dans cls_1 :
    compter_elements ( os . path . join ( folder_model_1 , 'train' , i ), False )
    compter_elements ( os . path . join ( folder_model_1 , 'val' , i ), False )
    compter_elements ( os . path . join ( folder_model_1 , 'test_1' , i ), False )

pour i dans cls_2 :
    compter_elements ( os.path.join ( folder_model_2 , 'train' , i ) , False ) compter_elements ( os.path.join ( folder_model_2
```

Nombre d'éléments : 1412

Nombre d'éléments : 354

```
Nombre d'éléments : 194
Nombre d'éléments : 1412
Nombre d'éléments : 354
Nombre d'éléments : 194
Nombre d'éléments : 1412
Nombre d'éléments : 354
Nombre d'éléments : 194
Nombre d'éléments : 1200
Nombre d'éléments : 300
Nombre d'éléments : 150
Nombre d'éléments : 1200
Nombre d'éléments : 300
Nombre d'éléments : 150
Nombre d'éléments : 1200
Nombre d'éléments : 300
Nombre d'éléments : 150
Nombre d'éléments : 1200
Nombre d'éléments : 300
Nombre d'éléments : 150
```

Dans [...

```
# En supposant que le répertoire 'test_1' existe conformément au code que vous avez fourni.
test_1_dir = folder_model_1 + '/test_1'

def get_random_defective_image ( test_dir ):
    """
        Renvoie le chemin d'accès à une image défectueuse aléatoire à partir du répertoire de test.
    """
    defective_dirs = [ d for d in os ..listdir ( test_dir ) if d == 'defective' and os .path .isdir ( os .path .join ( test_dir , 'defective' ) )
if not flawless_dirs :
    return None # Gère le cas où le répertoire 'defective' n'existe pas

    faulty_dir = os.path.join ( test_dir , faulty_dirs [ 0 ] ) images = [ f for f in os.listdir ( faulty_dir ) if os.path.isfile ( os .path .join ( faulty_dir , f ) )

    random_image = random .choice ( images )
    renvoie os .path .join ( faulty_dir , random_image )

image_test = obtenir_une_image_défectueuse_aléatoire ( test_1_dir )
image_test
```

```
Dehors[... '/model_1/test_1/defective/spotted_plum_116.png'
```

Notre architecture de dossier et donc bel est conçue avec 2 dossiers **model1 et model2** contenant chacun 3 dossiers **train, val et test** contenant eux aussi des classes d'images déjà équilibrées (defective, immature et unaffected) pour le model_1 et (tached, cracked, rotten et meurtri) pour le dossier model_2.

4) Conception du modèle

Passons maintenant à la création de notre architecture de classification d'image.

- **Objectif** : Utiliser 2 modèles dont l'un laisse la place à l'autre si il prédit une image comme étant 'défectueuse'
- **Modèle utilisé** : ConvNeXt v2
- **Métriques** : Exactitude, précision, rappel, f1_score, matrice de confusion
- **Régularisation** : Arrêt anticipé, lissage des étiquettes, abandon
- **Optimisation** : AdamW (Adaptive Moment Estimation with Weight Decay), décroissance du poids

4.1) Entraînement, validation et test

Dans [...

```
class Config :
    """
        Configuration pour ConvNeXtV2 : classification hiérarchique
    """

    MODEL1 = {
        'train' : folder_model_1 + '/train' ,
        'val' : folder_model_1 + '/val' ,
        'test' : folder_model_1 + '/test_1' ,
        'classes' : [ 'defective' , 'unaffected' , 'unripe' ],
        'num_classes' : 3
    }

    MODEL2 = {
        'train' : folder_model_2 + '/train' ,
        'val' : folder_model_2 + '/val' ,
        'test' : folder_model_2 + '/test_2' ,
        'classes' : [ 'taché' , 'fissuré' , 'meurtri' , 'pourri' ],
        'num_classes' : 4
    }
```

```
TAILLE_LOT    =  32
NOM_ÉPOQUES   =  15
LR   =  2e-5
PATIENCE     =  5
TAILLE_IMG   =  224
NOM_MODÈLE   = 'convnextv2_base.fcmae_ft_in1k'
TAUX_BAISSE   =  0,1
TAUX_BAISSE_ATTENTION =  0,1
TAUX_BAISSE_CHEMIN  =  0,1
DÉCROISSANCE_POIDS =  0,0001
LISSAGE_ÉTIQUETTE =  0,1
MOYENNE      = ( 0,485 ,  0,456 ,  0,406 )
Écart-type   = ( 0,229 ,  0,224 ,  0,225 )
```

Dans [...]

```
def get_dataloaders ( data_path , batch_size , img_size , mean , std ) :
    transform = transforms.Compose ( [ transforms.Resize ( ( img_size , img_size ) ) , transforms.ToTensor ( ) , transforms.Norm
```

Dans [...]

```
def build_convnext_model ( num_classes ):
    model = timm.create_model ( Config.MODEL_NAME , pretrained = True , num_classes = num_classes , drop_rate = Config.DROP_RATE
```

Dans [...]

```
def train_model ( model , train_loader , val_loader , num_epochs , save_path )
    device = torch.device ( "cuda" if torch.cuda.is_available ( ) else "cpu" ) model = model.to ( device ) optimizer = opt
```

```

meilleure_valeur_perte = float ('inf')
patience_counter = 0

pour l'époque dans la plage ( num_epochs ) :
    model.train ( ) train_loss , correct , total = 0 , 0 , 0

    pour les entrées , les cibles dans train_loader :
        entrées , cibles = entrées . à ( périphérique ), cibles . à ( périphérique )
        optimiseur . zero_grad ()
        sorties = modèle ( entrées )
        perte = critère ( sorties , cibles )
        perte . arrière ()
        optimiseur . étape ()

        train_loss += loss . item () * inputs . size ( 0 )
        _ , predicted = outputs . max ( 1 )
        correct += predicted . eq ( targets ) . sum () . item ()
        total += targets . size ( 0 )

    avg_train_loss = train_loss / total
    train_acc = correct / total

    val_loss , val_acc = evaluate_model ( modèle , val_loader )

    print ( f "Époque { epoch + 1 } / { num_epochs } , Perte de train : { avg_train_loss : .4f } , Acc du train : { train_acc : .4f } " )

    si val_loss < best_val_loss :
        best_val_loss = val_loss
        patience_counter = 0
        torch . save ( model . state_dict (), save_path )
    else :
        patience_counter += 1
        si patience_counter >= Config . PATIENCE :
            print ( "Arrêt anticipé" )
            break

```

Dans [...]

```

def evaluate_model ( model , loader )
    device = torch.device ( " cuda " if torch.cuda.is_available ( ) else " cpu " ) model.eval ( ) criteria = nn.CrossEntropyLoss ( )

```

Dans [...]

```
def test_model ( model , loader , class_names ) :
    device = torch.device ( " cuda " if torch.cuda.is_available ( ) else " cpu " ) model = model.to ( device ) model.eval ( )

    avec torch . no_grad ( ) :
        pour les entrées , les cibles dans le chargeur :
            inputs = inputs . to ( device )
            outputs = model ( inputs )
            _ , preds = outputs . max ( 1 )
            all_preds . extend ( preds . cpu ( ) . numpy ( ) )
            all_targets . extend ( targets . numpy ( ) )

    print ( " \n Rapport de classification : " )
    print ( classification_report ( all_targets , all_preds , target_names = class_names ) )

    cm = confusion_matrix ( all_targets , all_preds )
    plt.figure ( figsize = ( 8,6 ) ) sns.heatmap ( cm , annot = True , fmt = ' d ' , cmap = " Blues " , xticklabels = class_names ,
```

Dans [...]

```
def main () :
    # Modèle 1

    train_loader1 = get_dataloaders ( Config.MODEL1 [ ' train ' ] , Config.BATCH_SIZE , Config.IMG_SIZE , Config.MEAN , Config.S
```

```

model1 = build_convnext_model ( Config . MODEL1 [ 'num_classes' ] )
print ( "🚀 Entrainement du Modèle 1" )
train_model ( model1 , train_loader1 , val_loader1 , Config . NUM_EPOCHS , "model1_best.pt" )
model1 . load_state_dict ( torch . load ( "model1_best.pt" ) )
print ( "🚀 Test du Modèle 1" )
test_model ( model1 , test_loader1 , Config . MODEL1 [ 'classes' ] )

# Modèle 2
train_loader2 = get_dataloaders ( Config . MODEL2 [ 'train' ] , Config . BATCH_SIZE , Config . IMG_SIZE , Config . MEAN ,
val_loader2 = get_dataloaders ( Config . MODEL2 [ 'val' ] , Config . BATCH_SIZE , Config . IMG_SIZE , Config . MEAN ,
test_loader2 = get_dataloaders ( Config . MODEL2 [ 'test' ] , Config . BATCH_SIZE , Config . IMG_SIZE , Config . MEAN ,
C

model2 = build_convnext_model ( Config . MODEL2 [ 'num_classes' ] )
print ( "🚀 Entrainement du Model 2" )
train_model ( model2 , train_loader2 , val_loader2 , Config . NUM_EPOCHS , "model2_best.pt" )
model2 . load_state_dict ( torch . load ( "model2_best.pt" ) )
print ( "🚀 Test du Model 2" )
test_model ( model2 , test_loader2 , Config . MODEL2 [ 'classes' ] )

```

Dans [...]

```

si __name__ == '__main__':
    main ()

```

```

/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: Avertissement utilisateur :
Le secret `HF_TOKEN` n'existe pas dans vos secrets Colab.
Pour vous authentifier auprès du Hugging Face Hub, créez un jeton dans votre onglet paramètres (https://huggingface.co/settings/tokens), définissez-le comme secret dans votre Google Colab et redémarrez votre session.
Vous pourrez réutiliser ce secret dans tous vos cahiers.
Veuillez noter que l'authentification est recommandée mais toujours facultative pour accéder aux modèles ou ensembles de données publics.
    avertissements.warn(

```

🚀 Entrainement du Modèle 1

Époque 1/15, Perte de train : 0,7708, Acc du train : 0,7365, Perte de valeur : 0,4874, Acc de valeur : 0,8154
Époque 2/15, Perte de train : 0,5373, Acc du train : 0,8647, Perte de valeur : 0,4198, Acc de valeur : 0,8324
Époque 3/15, Perte de train : 0,4716, Acc du train : 0,9058, Perte de valeur : 0,3972, Acc de valeur : 0,8512
Époque 4/15, Perte de train : 0,4341, Acc du train : 0,9271, Perte de valeur : 0,4010, Acc de valeur : 0,8588
Époque 5/15, Perte de train : 0,4076, Acc du train : 0,9396, Perte de valeur : 0,4236, Acc de valeur : 0,8512
Époque 6/15, Perte de train : 0,3869, Acc du train : 0,9474, Perte de valeur : 0,4191, Acc de valeur : 0,8493
Époque 7/15, Perte de train : 0,3723, Acc du train : 0,9521, Perte de valeur : 0,4332, Acc de valeur : 0,8522

Époque 8/15, Perte de train : 0,3618, Acc du train : 0,9554, Perte de valeur : 0,4461, Acc de valeur : 0,8503

Arrêt anticipé

```
<ipython-input-42-cf889bc80828>:11: Avertissement : Vous utilisez `torch.load` avec `weights_only=False` (la valeur par défaut actuelle), qui utilise implicitement le module pickle par défaut. Il est possible de construire des données pickle malveillantes qui exécuteront du code arbitraire lors du dépickling (voir https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models pour plus de détails). Dans une prochaine version, la valeur par défaut de `weights_only` sera remplacée par `True`. Cela limite les fonctions exécutables lors du dépickling. Le chargement d'objets arbitraires via ce mode ne sera plus autorisé, sauf si l'on a explicitement ajouté à la liste blanche par l'utilisateur via `torch.serialization.add_safe_globals`. Nous vous recommandons de commencer par définir « weights_only=True » pour tout cas d'utilisation où vous n'avez pas le contrôle total du fichier chargé. Veuillez ouvrir un ticket sur GitHub pour tout problème lié à cette fonctionnalité expérimentale.
```

```
model1.load_state_dict(torch.load("model1_best.pt"))
```

🚀 Test du Model 1

Rapport de classification :

rappel de précision prise en charge du score f1

défectueux 0,83 0,88 0,85 194

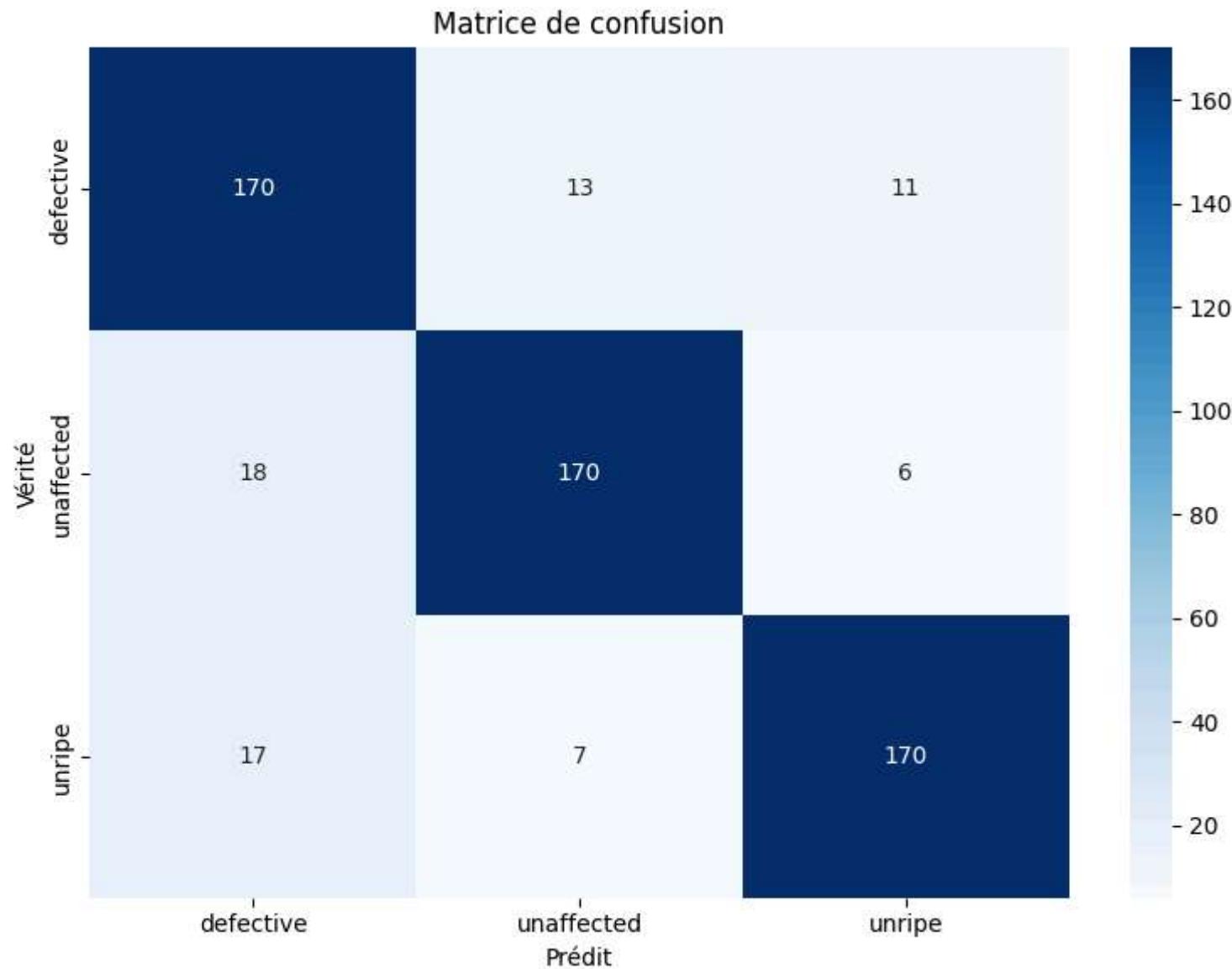
non affecté 0,89 0,88 0,89 194

pas mûr 0,91 0,88 0,89 194

précision 0,88 582

macro moyenne 0,88 0,88 0,88 582

moyenne pondérée 0,88 0,88 0,88 582



🚀 Entrainement du Model 2

Époque 1/15, Perte de train : 1,1091, Acc du train : 0,5965, Perte de valeur : 0,7700, Acc de valeur : 0,7308
 Époque 2/15, Perte de train : 0,7949, Acc du train : 0,7752, Perte de valeur : 0,5927, Acc de valeur : 0,7992
 Époque 3/15, Perte de train : 0,6318, Acc du train : 0,8629, Perte de valeur : 0,5365, Acc de valeur : 0,8033
 Époque 4/15, Perte de train : 0,5329, Acc du train : 0,9123, Perte de valeur : 0,5043, Acc de valeur : 0,8183
 Époque 5/15, Perte de train : 0,4831, Acc du train : 0,9423, Perte de valeur : 0,5357, Acc de valeur : 0,8067
 Époque 6/15, Perte de train : 0,4530, Acc du train : 0,9517, Perte de valeur : 0,5205, Acc de valeur : 0,8200

Époque 7/15, Perte de train : 0,4359, Acc du train : 0,9604, Perte de valeur : 0,5234, Acc de valeur : 0,8225
Époque 8/15, Perte de train : 0,4194, Acc du train : 0,9644, Perte de valeur : 0,5087, Acc de valeur : 0,8275
Époque 9/15, Perte de train : 0,4140, Acc du train : 0,9665, Perte de valeur : 0,5233, Acc de valeur : 0,8283
Arrêt anticipé

```
<ipython-input-42-cf889bc80828>:23: Avertissement : Vous utilisez `torch.load` avec `weights_only=False` (la valeur par défaut actuelle), qui utilise implicitement le module pickle par défaut. Il est possible de construire des données pickle malveillantes qui exécuteront du code arbitraire lors du dépickling (voir https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models pour plus de détails). Dans une prochaine version, la valeur par défaut de `weights_only` sera remplacée par `True`. Cela limite les fonctions exécutables lors du dépickling. Le chargement d'objets arbitraires via ce mode ne sera plus autorisé, sauf s'ils sont explicitement ajoutés à la liste blanche par l'utilisateur via `torch.serialization.add_safe_globals`. Nous vous recommandons de commencer par définir « weights_only=True » pour tout cas d'utilisation où vous n'avez pas le contrôle total du fichier chargé. Veuillez ouvrir un ticket sur GitHub pour tout problème lié à cette fonctionnalité expérimentale.
```

```
model2.load_state_dict(torch.load("model2_best.pt"))
```

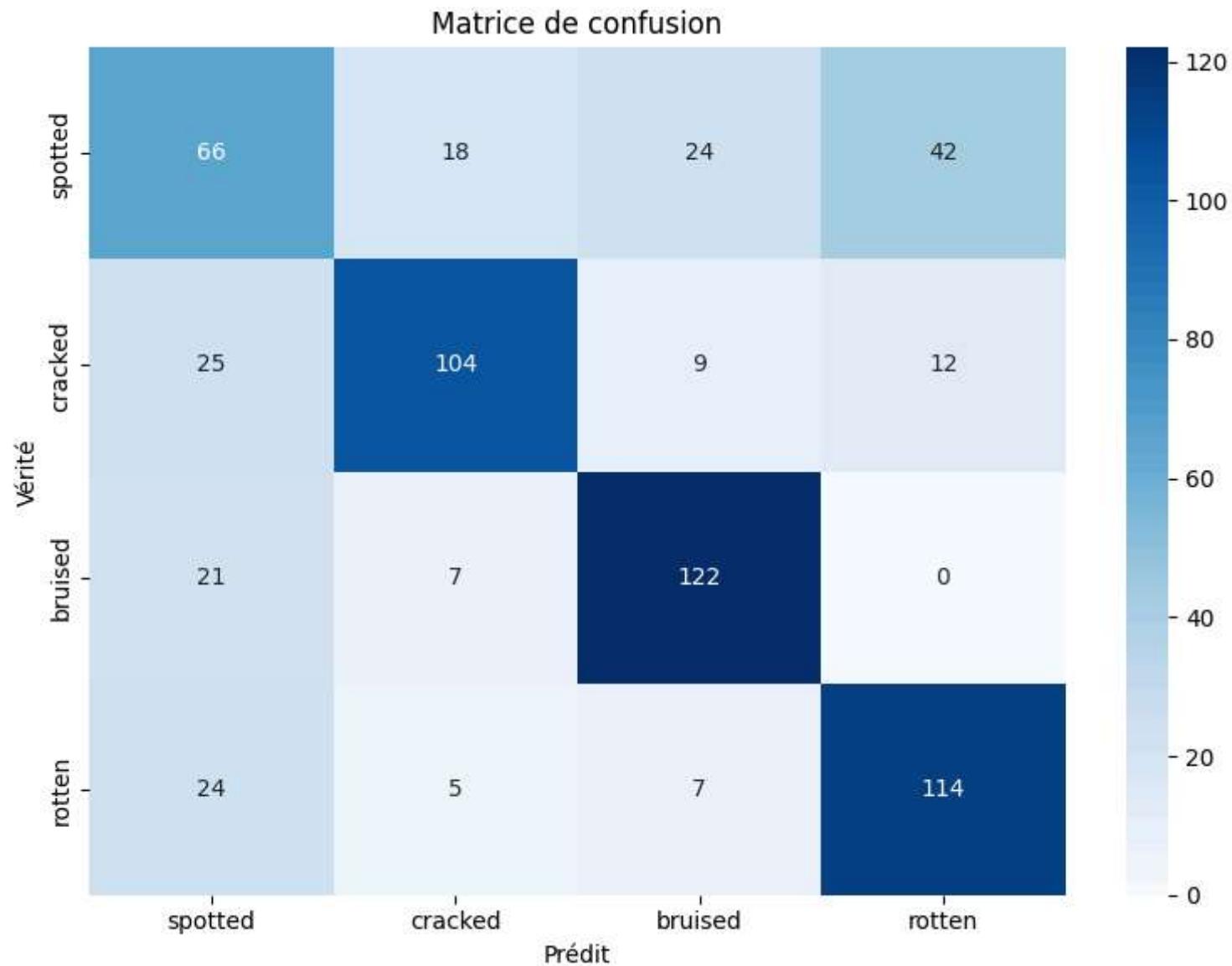
🚀 Test du Model 2

Rapport de classification :

rappel de précision prise en charge du score f1

```
repéré 0,49 0,44 0,46 150  
fissuré 0,78 0,69 0,73 150  
meurtri 0,75 0,81 0,78 150  
pourri 0,68 0,76 0,72 150
```

```
précision 0,68 600  
macro moyenne 0,67 0,68 0,67 600  
moyenne pondérée 0,67 0,68 0,67 600
```



Dans [...

```
image_test = obtenir_une_image_défectueuse_ aléatoire ( test_1_dir )  
image_test
```

Dehors[...]

```
'/model_1/test_1/defective/cracked_plum_38.png'
```

Dans [...
image_test

Dehors[...
'/model_1/test_1/defective/cracked_plum_38.png'

4.2) Configuration et essai pour une utilisation extérieure

Dans [...

```
# Classe de configuration Config_Pred :
MODEL_NAME = 'convnextv2_base.fcmae_ft_in1k'
IMG_SIZE = 224
MEAN = ( 0.485 , 0.456 , 0.406 )
STD = ( 0.229 , 0.224 , 0.225 )
MODEL1_CLASSES = [ 'défectueux' , 'non affecté' , 'pas mûr' ]
MODEL2_CLASSES = [ 'taché' , 'fissuré' , 'meurtri' , 'pourri' ]
MODEL1_NUM_CLASSES = 3
MODEL2_NUM_CLASSES = 4
```

Dans [...

```
# Fonction de chargement de modèle def load_model ( path , num_classes )
: model = create_model (
    Config_Pred.MODEL_NAME , pretrained = False , num_classes = num_classes ) model.load_state_dict ( torch.load ( path , map_
```

Dans [...

```
périphérique = torch . device ( "cuda" si torch . cuda . is_available () sinon "cpu" )

# Transformations
transform = transforms . Compose [
    transforms . Resize (( Config . IMG_SIZE , Config . IMG_SIZE )),
    transforms . ToTensor (),
    transforms . Normalize ( Config . MEAN , Config . STD )
])
```

```

# Fonction de prédiction hiérarchique
def predict_hierarchical ( image_path ) :
    # Charger image
    image = Image . ouvert ( chemin_image ) . convert ( "RGB" )
    img_tensor = transform ( image ) . décompresser ( 0 ) . à ( appareil )

    # Charger et prédire avec
    model1 model1 = load_model ( "model1_best.pt" , Config_Pred . MODEL1_NUM_CLASSES )
    with torch . no_grad (): output1 = model1 ( img_tensor ) pred1 = torch . argmax ( output1 , dim = 1 ) . item () class1 = Config_Pred . MODEL1_CLASSES [ pred1 ]

    # Si 'défectueux', prédire avec model2
    if class1 == "defective" :
        model2 = load_model ( "model2_best.pt" , Config_Pred . MODEL2_NUM_CLASSES )
        with torch . no_grad ():
            output2 = model2 ( img_tensor )
            pred2 = torch . argmax ( output2 , dim = 1 ) . item ()
            class2 = Config_Pred . MODEL2_CLASSES [ pred2 ]
        return { "level1" : class1 , "level2" : class2 }
    else :
        return { "level1" : class1 , "level2" : None }

```

Dans [...]

```

# Exemple d'utilisation
image_path = image_test
result = predict_hierarchical ( image_path )
print ( "Résultat hiérarchique:" , result )

```

<ipython-input-49-01e9201dca14>:8: Avertissement : Vous utilisez `torch.load` avec `weights_only=False` (la valeur par défaut actuelle), qui utilise implicitement le module pickle par défaut. Il est possible de construire des données pickle malveillantes qui exécuteront du code arbitraire lors du dépickling (voir <https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models> pour plus de détails). Dans une prochaine version, la valeur par défaut de `weights_only` sera remplacée par `True`. Cela limite les fonctions exécutables lors du dépickling. Le chargement d'objets arbitraires via ce mode ne sera plus autorisé, sauf s'ils sont explicitement ajoutés à la liste blanche par l'utilisateur via `torch.serialization.add_safe_globals`. Nous vous recommandons de commencer par définir « weights_only=True » pour tout cas d'utilisation où vous n'avez pas le contrôle total du fichier chargé. Veuillez ouvrir un ticket sur GitHub pour tout problème lié à cette fonctionnalité expérimentale.

```

    model.load_state_dict(torch.load(chemin, map_location=device))
Résultat hiérarchique : {'level1': 'defective', 'level2': 'cracked'}

```

4.3) Téléchargement des modèles en local

Dans [...

```
à partir des fichiers d'importation google.colab

# Si les modèles sont dans le dossier courant
files . télécharger ( "modell1_best.pt" )
fichiers . télécharger ( "modell2_best.pt" )
```

Dans [...

Dans [...