

Dans [7...]

```

# 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 [7...]

```

# 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 [7...]

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

2) Chargement des données

Dans [7...]

```

#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.path.join(path, 'african_plums')

# Obtenir les sous-dossiers

```

```

classes = [os.path.join(dossier_images, nom) for nom in os.listdir(dossier_images) if os.path.isdir(os.path.join(dossier_images, n

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

# Afficher une image de chaque dossier
for ax, dossier in zip(axes, classes[:6]): # Limiter à 6 dossiers
    # Charger la première image du dossier
    images = os.listdir(dossier)
    if images:
        image_path = os.path.join(dossier, images[55]) # Prendre la première image
        img = Image.open(image_path)
        ax.imshow(img)
        ax.axis('off') # Masquer les axes
        ax.set_title(os.path.basename(dossier)) # Titre avec le nom du dossier

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

```



In [75]:

```

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

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

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

df.sample(10)

```

Out[75]:

	Image ID	Label	Defect Type
2340	unaffected_plum_381	good	unaffected
163	bruised_plum_164	defective	bruised
4179	unripe_plum_499	unripe	unripe
4484	unripe_plum_804	unripe	unripe
3168	unaffected_plum_1209	good	unaffected
3346	unaffected_plum_1387	good	unaffected
1883	spotted_plum_683	defective	spotted
3551	unaffected_plum_1592	good	unaffected
254	bruised_plum_255	defective	bruised
4083	unripe_plum_403	unripe	unripe

In [76]:

`df.shape`

Out[76]:

(4507, 3)

In [77]:

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

Out[77]:

	Image_Id	Label	class
4148	unripe_plum_468	unripe	unripe
3794	unripe_plum_114	unripe	unripe
3865	unripe_plum_185	unripe	unripe
3510	unaffected_plum_1551	good	unaffected
1869	spotted_plum_669	defective	spotted
3983	unripe_plum_303	unripe	unripe
641	rotten_plum_161	defective	rotten

	Image_Id	Label	class
1774	spotted_plum_574	defective	spotted
2943	unaffected_plum_984	good	unaffected
4304	unripe_plum_624	unripe	unripe

In [78]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4507 entries, 0 to 4506
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          --          --      
 0   Image_Id    4507 non-null   object 
 1   Label        4507 non-null   object 
 2   class        4507 non-null   object 
dtypes: object(3)
memory usage: 105.8+ KB
```

In [79]: `df.isna().sum(axis=0)`

Out[79]:

0
Image_Id 0
Label 0
class 0

dtype: int64

In [80]: `df.duplicated().sum()
print(f"Nombre total de valeurs dupliquées : {df.duplicated().sum()}")`

Nombre total de valeurs dupliquées : 0

In [81]:

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

Nombre total d'image pour chaque type de classe :

Out[81]:

0

class
bruised 319
cracked 162
rotten 720
spotted 759
unaffected 1721
unripe 826

dtype: int64

In [82]:

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

Out[82]:

	Label	class
0	defective	[spotted, cracked, bruised, rotten]
1	good	[unaffected]
2	unripe	[unripe]

In [83]:

```
# 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
for nom in os.listdir(dossier_images):
    chemin_dossier = os.path.join(dossier_images, nom)
    if os.path.isdir(chemin_dossier):
```

```
images = [f for f in os.listdir(chemin_dossier) if f.lower().endswith('.png', '.jpg', '.jpeg', '.gif', '.bmp'))]
noms_dossiers.append(nom)
nombre_images.append(len(images))

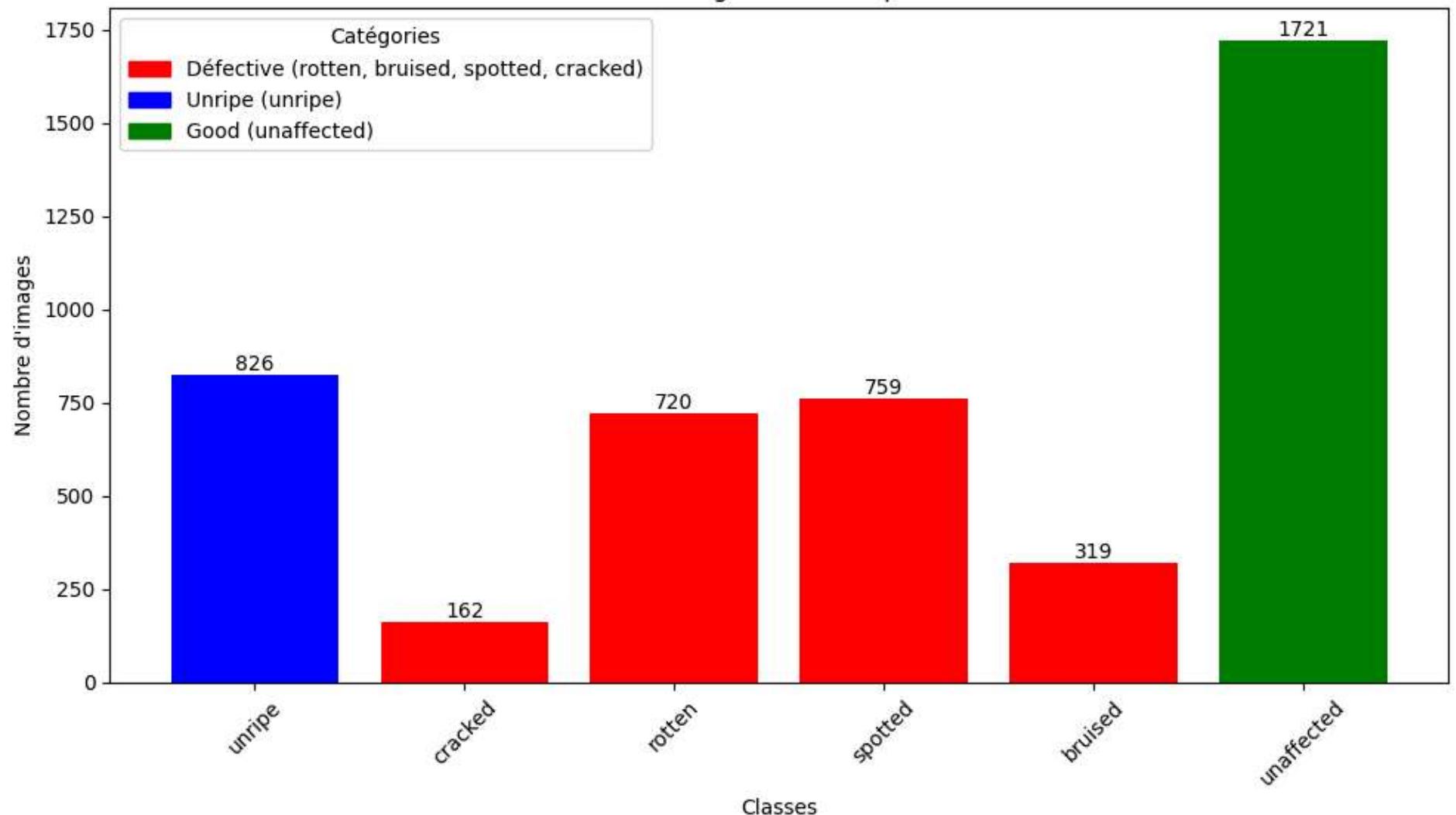
# Déterminer les couleurs des barres
couleurs = ['red' if nom in ['rotten', 'bruised', 'spotted', 'cracked'] else 'blue' if nom == 'unripe' else 'green' for nom in nom]

# 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.tight_layout() # Ajuster l'espacement
# Ajouter la légende
plt.legend(
    handles=[
        plt.Rectangle((0, 0), 1, 1, color='red', label='Défective (rotten, bruised, spotted, cracked)'),
        plt.Rectangle((0, 0), 1, 1, color='blue', label='Unripe (unripe)'),
        plt.Rectangle((0, 0), 1, 1, color='green', label='Good (unaffected)')
    ],
    title='Catégories'
)
plt.show()
```

Nombre d'images dans chaque dossier



En résumé, les classes **spotted**, **cracked**, **rotten** et **bruised** sont toutes parties de la classe **defective**, et la classe **good** se confond avec la classe **unaffected**. Il est aussi à noter que les données sont très déséquilibrées mais nous allons gérer cela dans les étapes suivantes avec **la data augmentation**.

3) Prétraitement des images

In [84]:

```
# Petite 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}")

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

In [85]:

```
compter_elements(path, True)
```

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

In [86]:

```
path
```

```
Out[86]: '/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 repertoire de travail (work_path) et y ajouter 2 dossiers **model_1** **et model_2** (folder_model_1 et folder_model_2) qui contiendront les ensembles de données de chaque model.
2. A l'intérieur de chaque dossier de model, 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). C-a-d 90% de spotted, 90% de enripe, ...
3. Consistant uniquement les dossiers test_1 et train_val_temp_1, on rassemble toutes les images contenues dans les dossiers **spotted, cracked, rotte, et bruised** dans un seul dossier du nom de **defective** afin d'avoir 3 classes.

4. Effectuer la data augmentation sur les données de test tout en équilibrant les différentes classes de sorte que les classes minoritaires aient le même nombre d'image que la classe majoritaire.
5. Effectuer aussi la data augmentation sur les dossier 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

In [87]:

```
# Définir les chemins
work_path = os.getcwd()
path_data = os.path.join(path, 'african_plums') # Dossier source
data_use = os.path.join(work_path, '/data') # Dossier cible
folder_model_1 = os.path.join(work_path, '/model_1')
folder_model_2 = os.path.join(work_path, '/model_2')
train_val_temp_1 = os.path.join(folder_model_1, 'train_val_temp_1')
train_val_temp_2 = os.path.join(folder_model_2, 'train_val_temp_2')
test_1 = os.path.join(folder_model_1, 'test_1')
test_2 = os.path.join(folder_model_2, 'test_2')
```

In [88]:

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

# Création des dossiers nécessaires
print("✍️ Crédit des dossiers requis...")
create_folder(data_use)
create_folder(folder_model_1)
create_folder(folder_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.path.exists(dst):
```

```

os.makedirs(dst)

items = os.listdir(src)
print(f"\nCopie de {len(items)} éléments de '{src}' vers '{dst}'...\n")

for item in tqdm(items, desc="Copie en cours", unit="élément"):
    s = os.path.join(src, item)
    d = os.path.join(dst, item)
    if os.path.isdir(s):
        shutil.copytree(s, d)
    else:
        shutil.copy2(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...

- trash Dossier existant supprimé : /data
- folder Nouveau dossier créé : /data
- trash Dossier existant supprimé : /model_1
- folder Nouveau dossier créé : /model_1
- trash Dossier existant supprimé : /model_2
- folder Nouveau dossier créé : /model_2
- folder Nouveau dossier créé : /model_1/train_val_temp_1
- folder Nouveau dossier créé : /model_2/train_val_temp_2
- folder Nouveau dossier créé : /model_1/test_1
- folder 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 [00:05<00:00, 1.09élément/s]

✓ Opération terminée avec succès ! Tous les fichiers ont été copiés 🎉

In [89]:

```

compter_elements(work_path, True)
compter_elements(data_use, True)
compter_elements(folder_model_1, True)
compter_elements(folder_model_2, True)

```

```
Éléments du répertoire : ['.config', 'sample_data']
Nombre d'éléments : 2
Éléments du répertoire : ['spotted', 'unripe', 'unaffected', 'cracked', 'rotten', 'bruised']
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

In [90]:

```
# Classes à traiter
classes_model_1 = ['spotted', 'cracked', 'bruised', 'unaffected', 'unripe', 'rotten']
classes_model_2 = ['spotted', 'cracked', 'bruised', 'rotten']

def split_images(source_root, target_train, target_test, classes, ratio=0.1):
    for class_name in 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(images)

        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
        for img in 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
        for img in 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, 9.44classe/s]
📁 Classe 'spotted': 684 entraînement/val | 75 test
📁 Classe 'cracked': 146 entraînement/val | 16 test
📁 Classe 'bruised': 288 entraînement/val | 31 test

🔍 Traitement des classes: 83%|██████████ | 5/6 [00:00<00:00, 6.45classe/s]
📁 Classe 'unaffected': 1549 entraînement/val | 172 test
📁 Classe 'unripe': 744 entraînement/val | 82 test

🔍 Traitement des classes: 100%|██████████ | 6/6 [00:00<00:00, 6.81classe/s]
📁 Classe 'rotten': 648 entraînement/val | 72 test
```

🚀 Séparation pour le model_2

```
🔍 Traitement des classes: 25%|██ | 1/4 [00:00<00:00, 7.31classe/s]
📁 Classe 'spotted': 684 entraînement/val | 75 test
📁 Classe 'cracked': 146 entraînement/val | 16 test

🔍 Traitement des classes: 75%|██████ | 3/4 [00:00<00:00, 13.68classe/s]
📁 Classe 'bruised': 288 entraînement/val | 31 test

🔍 Traitement des classes: 100%|██████████ | 4/4 [00:00<00:00, 11.41classe/s]
📁 Classe 'rotten': 648 entraînement/val | 72 test
```

✅ Séparation terminée avec succès pour les deux modèles ! 🎉

In [91]:

```
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 : ['spotted', 'unripe', 'unaffected', 'cracked', 'rotten', 'bruised']
 Nombre d'éléments : 6
 Éléments du répertoire : ['spotted', 'unripe', 'unaffected', 'cracked', 'rotten', 'bruised']
 Nombre d'éléments : 6
 Éléments du répertoire : ['spotted', 'cracked', 'rotten', 'bruised']
 Nombre d'éléments : 4
 Éléments du répertoire : ['spotted', 'cracked', 'rotten', 'bruised']
 Nombre d'éléments : 4

3.3) Création des classes 'defective'

In [92]:

```
# Dossiers à fusionner
defective_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):
    defective_path = os.path.join(root_dir, 'defective')
    os.makedirs(defective_path, exist_ok=True)

    # Liste dynamique des classes à fusionner à partir du contenu réel
    class_names = os.listdir(root_dir)
    for class_name in class_names:
        class_path = os.path.join(root_dir, class_name)

        # 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" Fusion '{class_name}' → 'defective' ({root_dir})", unit="img"):
                src = os.path.join(class_path, img)
                dst = os.path.join(defective_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(defective_path, f"{base}_{count}{ext}")
                        count += 1
```

```

shutil.move(src, dst)

# Supprimer Le dossier une fois fusion terminé
shutil.rmtree(class_path)
print(f"🗑️ 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 'spotted' → 'defective' (/model_1/train_val_temp_1): 100%|██████████| 684/684 [00:00<00:00, 35985.90img/s]
- 🗑️ Supprimé : /model_1/train_val_temp_1/spotted
- ✅ Conservé : unripe
- ✅ Conservé : unaffected

📦 Fusion 'cracked' → 'defective' (/model_1/train_val_temp_1): 100%|██████████| 146/146 [00:00<00:00, 29593.02img/s]

🗑️ Supprimé : /model_1/train_val_temp_1/cracked

📦 Fusion 'rotten' → 'defective' (/model_1/train_val_temp_1): 100%|██████████| 648/648 [00:00<00:00, 37032.77img/s]

🗑️ Supprimé : /model_1/train_val_temp_1/rotten

📦 Fusion 'bruised' → 'defective' (/model_1/train_val_temp_1): 100%|██████████| 288/288 [00:00<00:00, 33263.38img/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 'spotted' → 'defective' (/model_1/test_1): 100%|██████████| 75/75 [00:00<00:00, 26543.99img/s]
- 🗑️ Supprimé : /model_1/test_1/spotted
- ✅ Conservé : unripe
- ✅ Conservé : unaffected

📦 Fusion 'cracked' → 'defective' (/model_1/test_1): 100%|██████████| 16/16 [00:00<00:00, 20385.44img/s]

🗑️ Supprimé : /model_1/test_1/cracked

📦 Fusion 'rotten' → 'defective' (/model_1/test_1): 100%|██████████| 72/72 [00:00<00:00, 23119.73img/s]

```
Supprimé : /model_1/test_1/rotten
Fusion 'bruised' → 'defective' (/model_1/test_1): 100%|██████████| 31/31 [00:00<00:00, 22325.45img/s]
Supprimé : /model_1/test_1/bruised

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

✓ Fusion terminée avec succès pour model_1 ! 🎉
```

In [93]:

```
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 : ['unripe', 'unaffected', 'defective']
Nombre d'éléments : 3
Éléments du répertoire : ['unripe', 'unaffected', 'defective']
Nombre d'éléments : 3
Éléments du répertoire : ['spotted', 'cracked', 'rotten', 'bruised']
Nombre d'éléments : 4
Éléments du répertoire : ['spotted', 'cracked', 'rotten', 'bruised']
Nombre d'éléments : 4
```

In [94]:

```
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

In [95]:

```
def get_transforms_1():
    return transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomRotation(20),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
        transforms.RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0))
    ])
```

In [96]:

```
def get_transforms_2():
    return transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomRotation(20),
        transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.1), # Plus de variation
        transforms.RandomAffine(degrees=15, translate=(0.05, 0.05), shear=10), # Ajout d'un léger décalage et cisaillement
        transforms.RandomResizedCrop(size=(224, 224), scale=(0.7, 1.0))
    ])
```

Augmentation et équilibrage des dossiers du model 2

In [97]:

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

    class_counts = {}
    class_images = {}

    for cls in os.listdir(dataset_root):
        cls_path = os.path.join(dataset_root, cls)
        if not os.path.isdir(cls_path):
            continue
        images = [img for img in os.listdir(cls_path) if img.lower().endswith('.jpg', '.jpeg', '.png')]
        if not images:
            continue
        class_counts[cls] = len(images)
        class_images[cls] = images

    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 and max_count < 1500:
        target_count = 1500
    elif 'test' in dataset_root and 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 cible par classe : {target_count} (dans {dataset_root})")

    for cls in 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"✅ Classe '{cls}' déjà équilibrée ({current_count} images)")
            continue

        to_generate = target_count - current_count
        print(f"➡ Classe '{cls}' → {current_count} → {target_count} (Ajout de {to_generate} images)")

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

            try:
                with Image.open(img_path).convert("RGB") as 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
            except Exception as e:
                print(f"⚠️ Erreur sur {img_path} : {e}")
                continue

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

```

In [98]:

```
balance_dataset_with_augmentation(train_val_temp_2)
balance_dataset_with_augmentation(test_2)
```

📊 Nombre d'images cible par classe : 1500 (dans /model_2/train_val_temp_2)
⟳ Équilibrage en cours: 0% | 0/4 [00:00<?, ?classe/s]
⌚ Classe 'spotted' → 684 → 1500 (Ajout de 816 images)
⟳ Équilibrage en cours: 25%|██████ | 1/4 [01:02<03:08, 62.89s/classe]
⌚ Classe 'cracked' → 146 → 1500 (Ajout de 1354 images)
⟳ Équilibrage en cours: 50%|██████ | 2/4 [03:02<03:12, 96.40s/classe]
⌚ Classe 'rotten' → 648 → 1500 (Ajout de 852 images)
⟳ Équilibrage en cours: 75%|██████ | 3/4 [04:00<01:18, 78.59s/classe]
⌚ Classe 'bruised' → 288 → 1500 (Ajout de 1212 images)
⟳ Équilibrage en cours: 100%|██████ | 4/4 [05:44<00:00, 86.14s/classe]
✓ Équilibrage terminé pour : /model_2/train_val_temp_2

📊 Nombre d'images cible par classe : 150 (dans /model_2/test_2)
⟳ Équilibrage en cours: 0% | 0/4 [00:00<?, ?classe/s]
⌚ Classe 'spotted' → 75 → 150 (Ajout de 75 images)
⟳ Équilibrage en cours: 25%|██████ | 1/4 [00:04<00:13, 4.64s/classe]
⌚ Classe 'cracked' → 16 → 150 (Ajout de 134 images)
⟳ Équilibrage en cours: 50%|██████ | 2/4 [00:13<00:14, 7.16s/classe]
⌚ Classe 'rotten' → 72 → 150 (Ajout de 78 images)
⟳ Équilibrage en cours: 75%|██████ | 3/4 [00:18<00:06, 6.07s/classe]
⌚ Classe 'bruised' → 31 → 150 (Ajout de 119 images)
⟳ Équilibrage en cours: 100%|██████ | 4/4 [00:26<00:00, 6.60s/classe]
✓ Équilibrage terminé pour : /model_2/test_2

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

In [99]:

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

    class_counts = {}
    class_images = {}
```

```
for cls in os.listdir(input_dir):
    cls_path = os.path.join(input_dir, cls)
    if not os.path.isdir(cls_path):
        continue
    images = [img for img in os.listdir(cls_path) if img.lower().endswith(('.jpg', '.jpeg', '.png'))]
    if not images:
        continue
    class_counts[cls] = len(images)
    class_images[cls] = images

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 représentée initialement dans {input_dir} : {max_count} images")
print(f"🎯 Nombre cible d'images par classe dans {input_dir} : {target_count} images")

for cls in 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

    if current_count >= target_count and 'train_val_temp' not in 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 [01:02<00:00, 20.95s/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 [00:06<00:00, 2.16s/classe]

✅ Classe 'defective' déjà équilibrée (194 images)

✅ Data augmentation terminée pour /model_1/test_1 🎉

In [100...]

```
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)
```

```

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))

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

```

🌟 Application aux deux modèles

```

split_train_val(train_val_temp_1, folder_model_1)
split_train_val(train_val_temp_2, folder_model_2)

```

📁 unripe → train: 100% | ██████████ | 1412/1412 [00:00<00:00, 39981.89img/s]
 📁 unripe → val: 100% | ██████████ | 354/354 [00:00<00:00, 40291.54img/s]
 📁 unaffected → train: 100% | ██████████ | 1412/1412 [00:00<00:00, 31679.85img/s]
 📁 unaffected → val: 100% | ██████████ | 354/354 [00:00<00:00, 41323.19img/s]
 📁 defective → train: 100% | ██████████ | 1412/1412 [00:00<00:00, 38773.09img/s]
 📁 defective → val: 100% | ██████████ | 354/354 [00:00<00:00, 39267.52img/s]

☒ Suppression du dossier temporaire : /model_1/train_val_temp_1

✅ Split terminé pour /model_1 ! 🎉

📁 spotted → train: 100% | ██████████ | 1200/1200 [00:00<00:00, 41589.87img/s]
 📁 spotted → val: 100% | ██████████ | 300/300 [00:00<00:00, 37665.50img/s]
 📁 cracked → train: 100% | ██████████ | 1200/1200 [00:00<00:00, 40358.30img/s]
 📁 cracked → val: 100% | ██████████ | 300/300 [00:00<00:00, 40094.68img/s]
 📁 rotten → train: 100% | ██████████ | 1200/1200 [00:00<00:00, 39682.46img/s]
 📁 rotten → val: 100% | ██████████ | 300/300 [00:00<00:00, 40490.77img/s]

```
📁 bruised → train: 100%|██████████| 1200/1200 [00:00<00:00, 40660.21img/s]
📁 bruised → val: 100%|██████████| 300/300 [00:00<00:00, 40584.80img/s]
🗂 Suppression du dossier temporaire : /model_2/train_val_temp_2
✅ Split terminé pour /model_2 ! 🎉
```

In [101...]

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

Out[101...]

```
"Le chemin spécifié n'existe pas."
```

In [102...]

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

Out[102...]

```
"Le chemin spécifié n'existe pas."
```

In [103...]

```
compter_elements(folder_model_1, True)
compter_elements(folder_model_1 + '/train', True)
compter_elements(folder_model_1 + '/val', True)
compter_elements(folder_model_1 + '/test_1', True)
compter_elements(folder_model_2, True)
compter_elements(folder_model_2 + '/train', True)
compter_elements(folder_model_2 + '/val', True)
compter_elements(folder_model_2 + '/test_2', True)
```

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

```
Nombre d'éléments : 3
```

```
Éléments du répertoire : ['unripe', 'unaffected', 'defective']
```

```
Nombre d'éléments : 3
```

```
Éléments du répertoire : ['unripe', 'unaffected', 'defective']
```

```
Nombre d'éléments : 3
```

```
Éléments du répertoire : ['unripe', 'unaffected', 'defective']
```

```
Nombre d'éléments : 3
```

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

```
Nombre d'éléments : 3
```

```
Éléments du répertoire : ['spotted', 'cracked', 'rotten', 'bruised']
```

```
Nombre d'éléments : 4
```

```
Éléments du répertoire : ['spotted', 'cracked', 'rotten', 'bruised']
```

```
Nombre d'éléments : 4
```

```
Éléments du répertoire : ['spotted', 'cracked', 'rotten', 'bruised']
```

```
Nombre d'éléments : 4
```

In [104...]

```
# Vérifions si les ensembles de données sont équilibrés

cls_1 = ['unaffected', 'defective', 'unripe']
cls_2 = ['spotted', 'cracked', 'bruised', 'rotten']
for i in 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)

for i in cls_2:
    compter_elements(os.path.join(folder_model_2, 'train', i), False)
    compter_elements(os.path.join(folder_model_2, 'val', i), False)
    compter_elements(os.path.join(folder_model_2, 'test_2', i), False)
```

```
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
```

In [105...]

```
# Assuming 'test_1' directory exists as per your provided code.
test_1_dir = folder_model_1 + '/test_1'

def get_random_defective_image(test_dir):
    """
    Returns the path to a random defective image from the test directory.
```

```

"""
defective_dirs = [d for d in os.listdir(test_dir) if d == 'defective' and os.path.isdir(os.path.join(test_dir,d))]
if not defective_dirs:
    return None # Handle the case where 'defective' directory doesn't exist

defective_dir = os.path.join(test_dir, defective_dirs[0])
images = [f for f in os.listdir(defective_dir) if os.path.isfile(os.path.join(defective_dir, f))]
if not images:
    return None # Handle the case where the 'defective' directory is empty

random_image = random.choice(images)
return os.path.join(defective_dir, random_image)

image_test = get_random_defective_image(test_1_dir)
image_test

```

Out[105...]

'/model_1/test_1/defective/bruised_plum_100.png'

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

4) Conception du model

Maintenant passons à la création de notre architecture de classification d'image.

- **Objectif** : Utiliser 2 models dont l'un laisse la place à l'autre si il prédit une image comme étant 'defective'
- **Modele utilisé** : ViT (**Vision Tranformers**)
- **Métriques** : Accuracy, precision, recall, f1_score, matrice de confusion
- **Régularisation** : Early stopping, label smoothing, dropout
- **Optimisation** : AdamW (Adaptive Moment Estimation with Weight Decay), weight decay

4.1) Entrainement, validation et test

In [106...]

```

class Config:
"""
Configuration pour les modèles de classification hiérarchique.
Définit les chemins des données, les noms des classes et les hyperparamètres d'entraînement.

```

```

"""
# Paths pour model_1 (classification de niveau supérieur en 3 classes)
MODEL1 = {
    'train': folder_model_1 + '/train',
    'val': folder_model_1 + '/val',
    'test': folder_model_1 + '/test_1',
    'classes': ['defective', 'unaffected', 'unripe'],
    'num_classes': 3
}

# Paths pour model_2 (classification de niveau inférieur en 4 classes, pour les cas 'defective')
MODEL2 = {
    'train': folder_model_2 + '/train',
    'val': folder_model_2 + '/val',
    'test': folder_model_2 + '/test_2',
    'classes': ['spotted', 'cracked', 'bruised', 'rotten'],
    'num_classes': 4
}

# Paramètres communs
BATCH_SIZE = 32
NUM_EPOCHS = 25
LR = 2e-5
PATIENCE = 5
IMG_SIZE = 224
MODEL_NAME = 'vit_base_patch16_224'
DROP_RATE = 0.1
ATTENTION_DROP_RATE = 0.1
DROP_PATH_RATE = 0.1
WEIGHT_DECAY = 0.0001
LABEL_SMOOTHING = 0.1
MEAN = (0.485, 0.456, 0.406)
STD = (0.229, 0.224, 0.225)

```

In [107...]

```

def get_dataloaders(data_path, batch_size, img_size, mean, std):
    transform = transforms.Compose([
        transforms.Resize((img_size, img_size)),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ])
    dataset = datasets.ImageFolder(data_path, transform=transform)
    dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
    return dataloader

```

In [108...]

```
def build_vit_model(num_classes):
    model = timm.create_model(
        Config.MODEL_NAME,
        pretrained=True,
        num_classes=num_classes,
        drop_rate=Config.DROP_RATE,
        drop_path_rate=Config.DROP_PATH_RATE,
        attn_drop_rate=Config.ATTENTION_DROP_RATE
    )
    return model
```

In [109...]

```
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 = optim.AdamW(model.parameters(), lr=Config.LR, weight_decay=Config.WEIGHT_DECAY)
    criterion = nn.CrossEntropyLoss(label_smoothing=Config.LABEL_SMOOTHING)
    best_val_loss = float('inf')
    patience_counter = 0

    for epoch in range(num_epochs):
        model.train()
        train_loss, correct, total = 0, 0, 0
        for inputs, targets in train_loader:
            inputs, targets = inputs.to(device), targets.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            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(model, val_loader)
```

```
print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {avg_train_loss:.4f}, Train Acc: {train_acc:.4f}, Val Loss: {val_loss:.4f}\n")

if val_loss < best_val_loss:
    best_val_loss = val_loss
    patience_counter = 0
    torch.save(model.state_dict(), save_path)
else:
    patience_counter += 1
    if patience_counter >= Config.PATIENCE:
        print("Early stopping")
        break
```

In [110...]

```
def evaluate_model(model, loader):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.eval()
    criterion = nn.CrossEntropyLoss()
    total_loss, correct, total = 0, 0, 0
    with torch.no_grad():
        for inputs, targets in loader:
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            total_loss += loss.item() * inputs.size(0)
            _, predicted = outputs.max(1)
            correct += predicted.eq(targets).sum().item()
            total += targets.size(0)
    return total_loss / total, correct / total
```

In [111...]

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

    all_preds, all_targets = [], []

    with torch.no_grad():
        for inputs, targets in loader:
            inputs = inputs.to(device)
            outputs = model(inputs)
            _, preds = outputs.max(1)
            all_preds.extend(preds.cpu().numpy())
            all_targets.extend(targets.numpy())
```

```
# Rapport texte
print("\nClassification Report:")
print(classification_report(all_targets, all_preds, target_names=class_names))

# Matrice de confusion
cm = confusion_matrix(all_targets, all_preds)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.title("Matrice de confusion")
plt.xlabel("Prédit")
plt.ylabel("Vérité")
plt.tight_layout()
plt.show()
```

In [112...]

```
def main():
    # Model 1

    train_loader1 = get_dataloaders(Config.MODEL1['train'], Config.BATCH_SIZE, Config.IMG_SIZE, Config.MEAN, Config.STD)
    val_loader1 = get_dataloaders(Config.MODEL1['val'], Config.BATCH_SIZE, Config.IMG_SIZE, Config.MEAN, Config.STD)
    test_loader1 = get_dataloaders(Config.MODEL1['test'], Config.BATCH_SIZE, Config.IMG_SIZE, Config.MEAN, Config.STD)

    model1 = build_vit_model(Config.MODEL1['num_classes'])
    print("🚀 Entrainement du Model 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 Model 1")
    test_model(model1, test_loader1, Config.MODEL1['classes'])

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

    model2 = build_vit_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'])
```

In [113...]

```
if __name__ == '__main__':
    main()
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.

  warnings.warn(
```

🚀 Entrainement du Model 1

```
Epoch 1/25, Train Loss: 0.7328, Train Acc: 0.7460, Val Loss: 0.4178, Val Acc: 0.8352
Epoch 2/25, Train Loss: 0.5371, Train Acc: 0.8657, Val Loss: 0.3785, Val Acc: 0.8701
Epoch 3/25, Train Loss: 0.4822, Train Acc: 0.9004, Val Loss: 0.4023, Val Acc: 0.8522
Epoch 4/25, Train Loss: 0.4436, Train Acc: 0.9266, Val Loss: 0.3886, Val Acc: 0.8606
Epoch 5/25, Train Loss: 0.4174, Train Acc: 0.9341, Val Loss: 0.3616, Val Acc: 0.8766
Epoch 6/25, Train Loss: 0.3990, Train Acc: 0.9438, Val Loss: 0.3976, Val Acc: 0.8682
Epoch 7/25, Train Loss: 0.3948, Train Acc: 0.9471, Val Loss: 0.4120, Val Acc: 0.8653
Epoch 8/25, Train Loss: 0.3823, Train Acc: 0.9525, Val Loss: 0.4237, Val Acc: 0.8522
Epoch 9/25, Train Loss: 0.3763, Train Acc: 0.9533, Val Loss: 0.3966, Val Acc: 0.8701
Epoch 10/25, Train Loss: 0.3724, Train Acc: 0.9509, Val Loss: 0.4128, Val Acc: 0.8588
```

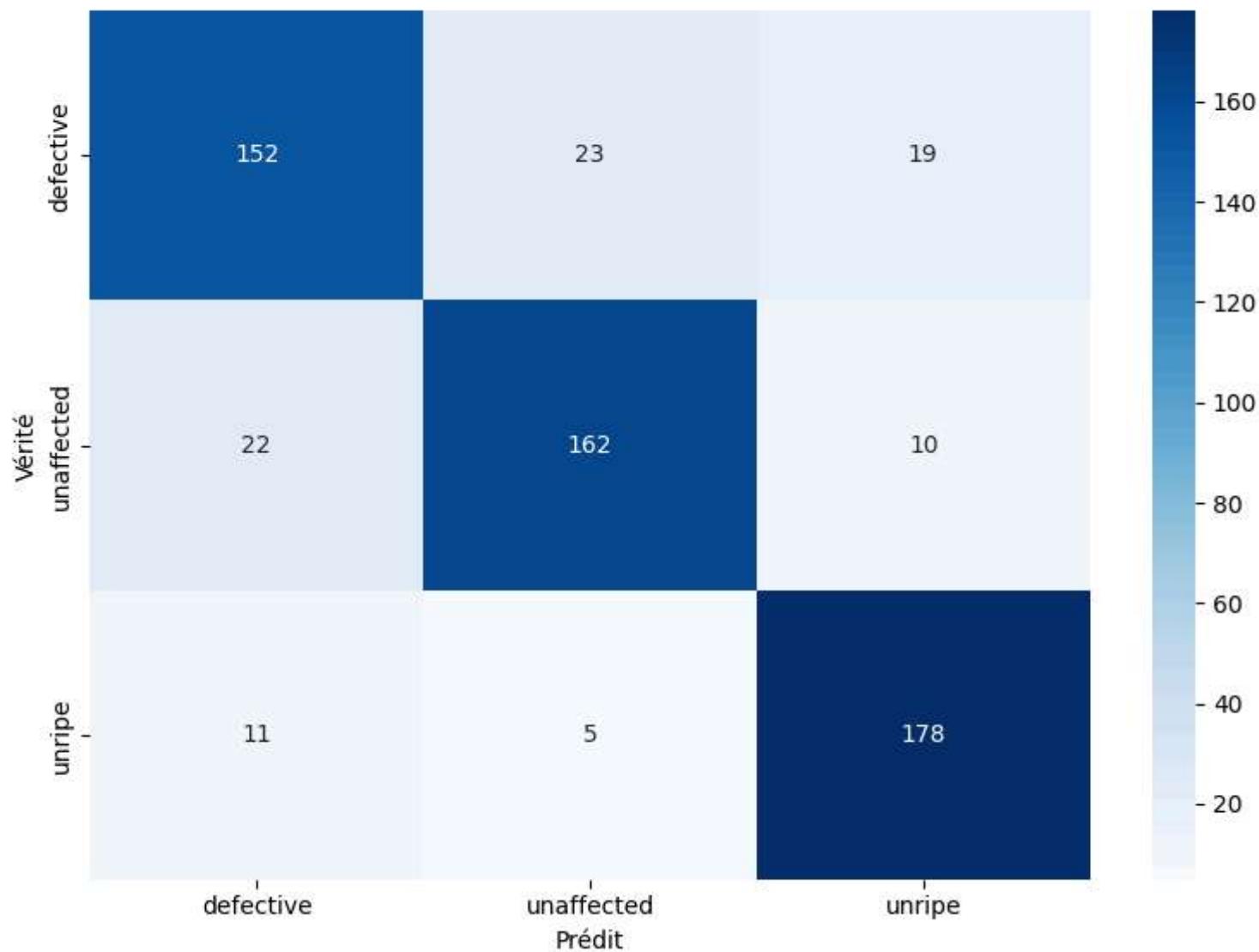
Early stopping

🚀 Test du Model 1

Classification Report:

	precision	recall	f1-score	support
defective	0.82	0.78	0.80	194
unaffected	0.85	0.84	0.84	194
unripe	0.86	0.92	0.89	194
accuracy			0.85	582
macro avg	0.84	0.85	0.84	582
weighted avg	0.84	0.85	0.84	582

Matrice de confusion



🚀 Entrainement du Model 2

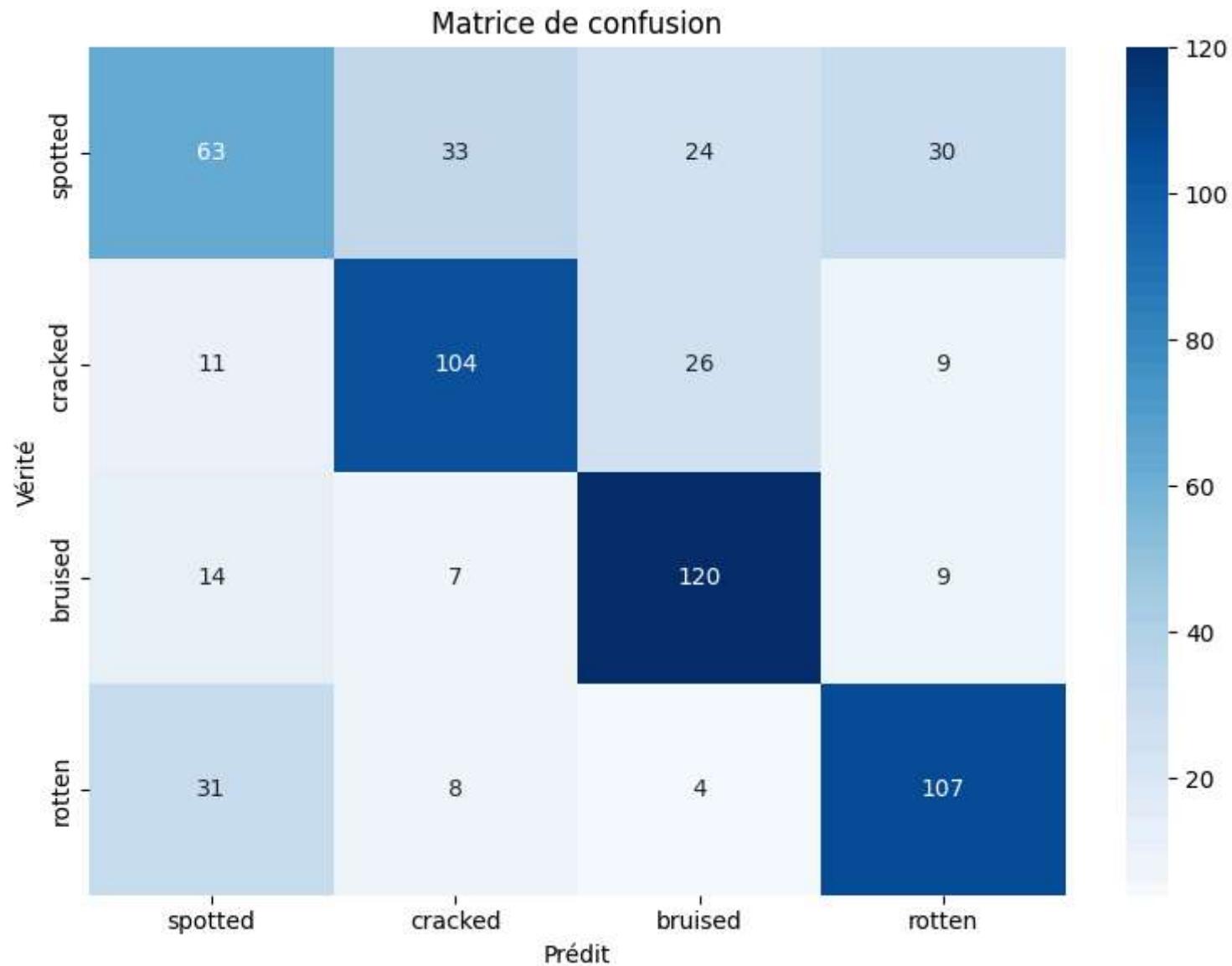
Epoch 1/25, Train Loss: 1.1165, Train Acc: 0.5558, Val Loss: 0.7543, Val Acc: 0.7000
Epoch 2/25, Train Loss: 0.8138, Train Acc: 0.7550, Val Loss: 0.6857, Val Acc: 0.7342
Epoch 3/25, Train Loss: 0.6628, Train Acc: 0.8413, Val Loss: 0.6059, Val Acc: 0.7750
Epoch 4/25, Train Loss: 0.5469, Train Acc: 0.9142, Val Loss: 0.5892, Val Acc: 0.7800
Epoch 5/25, Train Loss: 0.5069, Train Acc: 0.9337, Val Loss: 0.5497, Val Acc: 0.7925
Epoch 6/25, Train Loss: 0.4659, Train Acc: 0.9535, Val Loss: 0.5825, Val Acc: 0.7908

Epoch 7/25, Train Loss: 0.4429, Train Acc: 0.9650, Val Loss: 0.5529, Val Acc: 0.8133
Epoch 8/25, Train Loss: 0.4332, Train Acc: 0.9648, Val Loss: 0.5372, Val Acc: 0.8217
Epoch 9/25, Train Loss: 0.4240, Train Acc: 0.9694, Val Loss: 0.5644, Val Acc: 0.8133
Epoch 10/25, Train Loss: 0.4204, Train Acc: 0.9702, Val Loss: 0.6265, Val Acc: 0.7900
Epoch 11/25, Train Loss: 0.4211, Train Acc: 0.9665, Val Loss: 0.5741, Val Acc: 0.8050
Epoch 12/25, Train Loss: 0.4207, Train Acc: 0.9681, Val Loss: 0.5879, Val Acc: 0.8050
Epoch 13/25, Train Loss: 0.4163, Train Acc: 0.9669, Val Loss: 0.5959, Val Acc: 0.8025
Early stopping

🚀 Test du Model 2

Classification Report:

	precision	recall	f1-score	support
spotted	0.53	0.42	0.47	150
cracked	0.68	0.69	0.69	150
bruised	0.69	0.80	0.74	150
rotten	0.69	0.71	0.70	150
accuracy			0.66	600
macro avg	0.65	0.66	0.65	600
weighted avg	0.65	0.66	0.65	600



In [121...]

```
image_test = get_random_defective_image(test_1_dir)
image_test
```

Out[121...]

```
'/model_1/test_1/defective/spotted_plum_216.png'
```

```
In [122... image_test
```

```
Out[122... '/model_1/test_1/defective/spotted_plum_216.png'
```

4.2) Configuration et essai pour une utilisation extérieur

```
In [123... # Configuration
```

```
class Config_Pred:  
    MODEL_NAME = 'vit_base_patch16_224'  
    IMG_SIZE = 224  
    MEAN = (0.485, 0.456, 0.406)  
    STD = (0.229, 0.224, 0.225)  
    MODEL1_CLASSES = ['defective', 'unaffected', 'unripe']  
    MODEL2_CLASSES = ['spotted', 'cracked', 'bruised', 'rotten']  
    MODEL1_NUM_CLASSES = 3  
    MODEL2_NUM_CLASSES = 4
```

```
In [124... # 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_location=device))  
    model = model.to(device)  
    model.eval()  
    return model
```

```
In [125... device = torch.device("cuda" if torch.cuda.is_available() else "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.open(image_path).convert("RGB")
    img_tensor = transform(image).unsqueeze(0).to(device)

    # 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 'defective', 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}
```

In [126...]

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

Résultat hiérarchique: {'level1': 'unripe', 'level2': None}

4.3) Téléchargement des models en local

In [127...]

```
from google.colab import files

# Si les modèles sont dans le dossier courant
files.download("model1_best.pt")
#files.download("model2_best.pt")
```