Documentation du Workflow et de la Pipeline pour la Classification d'Images de Prunes Africaines

Cette documentation décrit en détail le workflow, le dataflow, le flowchart, l'architecture globale, et un scénario de test en production pour un code qui constitue une pipeline complète d'entraînement d'un modèle de classification d'images de prunes africaines. Ce pipeline inclut des étapes de prétraitement, de fine-tuning, d'évaluation et de test.

---

1. **Workflow (Flux de travail)**

Le workflow décrit les étapes principales de l'exécution du programme, du début à la fin.

1. **Chargement des données** :
    - Le programme commence par charger un fichier pickle (**image_data_with_images.pkl**) contenant un dataframe avec des images (colonne **Image**), des étiquettes (**Label** : unaffected, unripe, defective), et des types de défauts pour les images défectueuses (**Defect Type** : bruised, cracked, etc.).
2. **Préparation des données** (**prepare_data**) :
    - **Redimensionnement initial** : Les images sont redimensionnées à une taille spécifiée (**resize_height** x **resize_width**, par défaut 512x512) via la fonction **resize_images**.
    - **Suppression de l'arrière-plan** : Utilisation de **rembg** pour isoler les prunes africaines en supprimant l'arrière-plan.
    - **Augmentation des données** : Utilisation d'Albumentations pour équilibrer les classes en générant des images augmentées (rotations, flips, bruit, etc.) afin que chaque classe ait 1,721 images.
    - Un nouveau dataframe **augmented_df** est créé avec les images originales et augmentées.
3. **Affichage des images** (**display_n_images**) :
    - Affiche 10 images aléatoires (nombre configurable via **n**) après la préparation des données, avec leurs étiquettes (**Label** et **Defect Type**).
4. **Création du dataset et des DataLoaders** :
    - Un dataset personnalisé (**AfricanPlumsFromDataFrame**) est créé à partir du dataframe préparé.
    - Les données sont divisées en ensembles d'entraînement (70%), de validation (15%), et de test (15%) avec stratification.
    - Des DataLoaders PyTorch sont créés pour chaque ensemble (train, val, test).
5. **Création et configuration des modèles** (**create_finetune_models**) :
    - Deux modèles pré-entraînés (choisis via **HYPERPARAMS['model_name']**, par défaut ResNet18) sont créés : un pour la classification globale (**global_model**) et un pour la classification des défauts (**defect_model**).

- Les couches fully connected sont remplacées par une architecture personnalisée (**CustomFC**) avec des hyperparamètres configurables (nombre de couches, taille des couches, dropout).

6. **Entraînement et validation** (**train_and_validate**) :

   - Entraînement des deux modèles sur l'ensemble d'entraînement, avec validation sur l'ensemble de validation à chaque époque.
   - Calcul des pertes et des précisions pour les deux modèles (global et défauts).
   - Les métriques sont enregistrées dans un historique pour générer des courbes d'apprentissage.

7. **Visualisation des courbes d'apprentissage** (**plot_training_history**) :
   - Affiche et sauvegarde des graphiques de perte et de précision (entraînement et validation) pour les deux modèles.

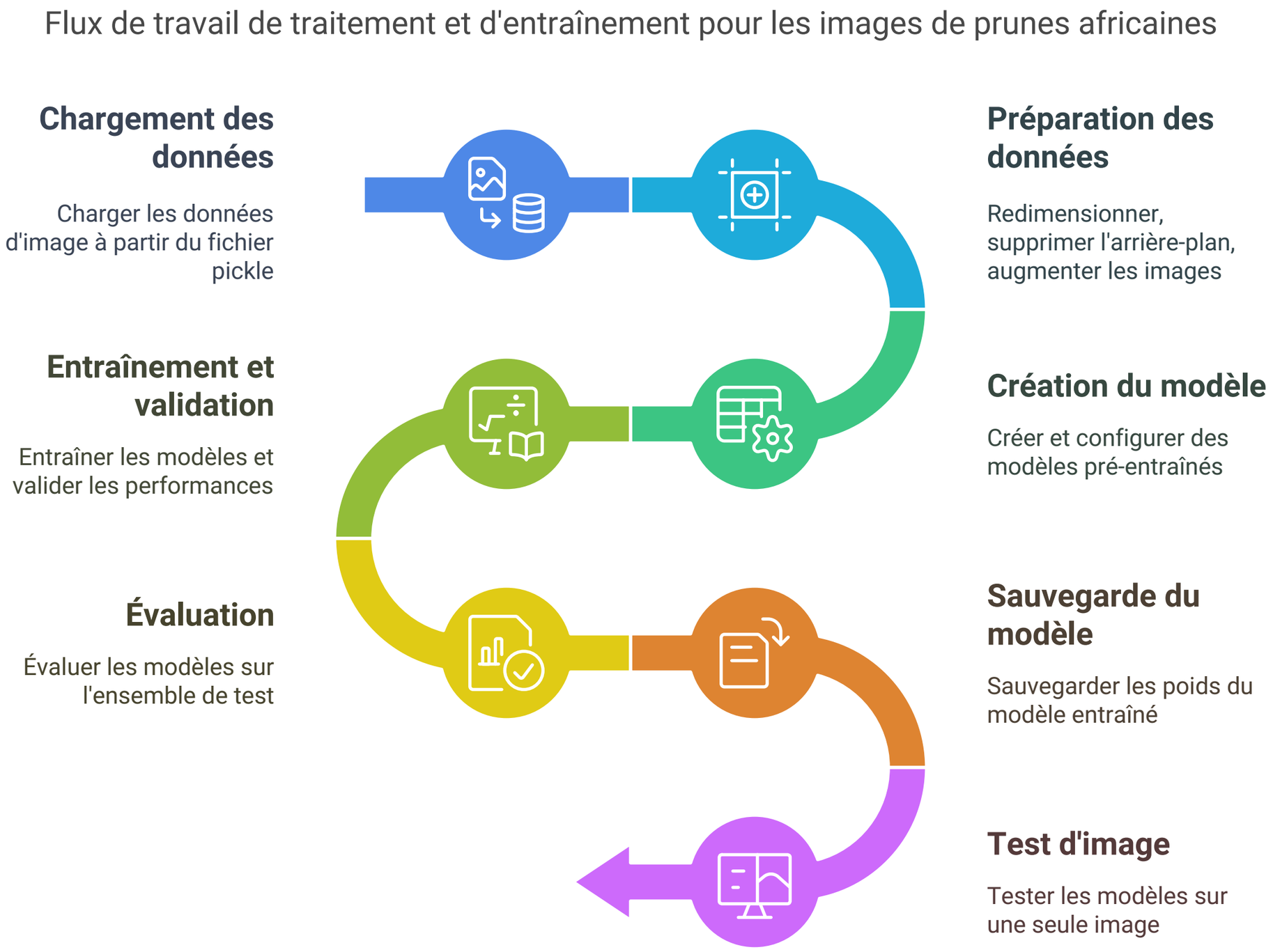
8. **Évaluation sur l'ensemble de test** (**evaluate_model**) :

   - Évalue les modèles sur l'ensemble de test et génère des rapports de classification (précision, rappel, F1-score) pour chaque classe.

9. **Sauvegarde des modèles** :

   - Les poids des modèles sont sauvegardés (par exemple, **global_model_finetuned_resnet18.pth**).

10. **Test sur une image** (**test_model_on_image**) :
    - Teste les modèles sur une image de l'ensemble préparé et affiche les prédictions (globale et défaut).



---

2. **Dataflow (Flux des données)**

Le dataflow décrit comment les données circulent à travers le pipeline.

1. **Entrée : Dataframe image_data** :
    - Colonnes : **Image** (tableau NumPy, format C,H,W), **Label**, **Defect Type**, **Image ID**.
    - Exemple : Une image de prune africaine avec **Label="defective"**, **Defect Type="bruised"**.
2. **Préparation des données** (**prepare_data**) :
    - **Redimensionnement** : Les images sont redimensionnées à **resize_height** x **resize_width** (par défaut 512x512).
        - Entrée : Image (C,H,W)
        - Sortie : Image redimensionnée (C,512,512)
    - **Suppression de l'arrière-plan** : Utilisation de **rembg** pour isoler la prune.
        - Entrée : Image redimensionnée (C,512,512)
        - Sortie : Image sans arrière-plan (C,512,512)

- **Augmentation** : Albumentations génère des images supplémentaires pour équilibrer les classes.
    - Entrée : Image sans arrière-plan (C,512,512)
    - Sortie : Nouvelles images augmentées (C,512,512)
- **Résultat** : Dataframe **augmented_df** avec des images prétraitées et augmentées.

3. **Affichage des images** (**display_n_images**) :
    - Sélectionne **n** images aléatoires de **augmented_df**.
    - Redimensionne les images à **image_size** (par défaut 224x224) pour l'affichage.
    - Affiche les images avec leurs étiquettes.

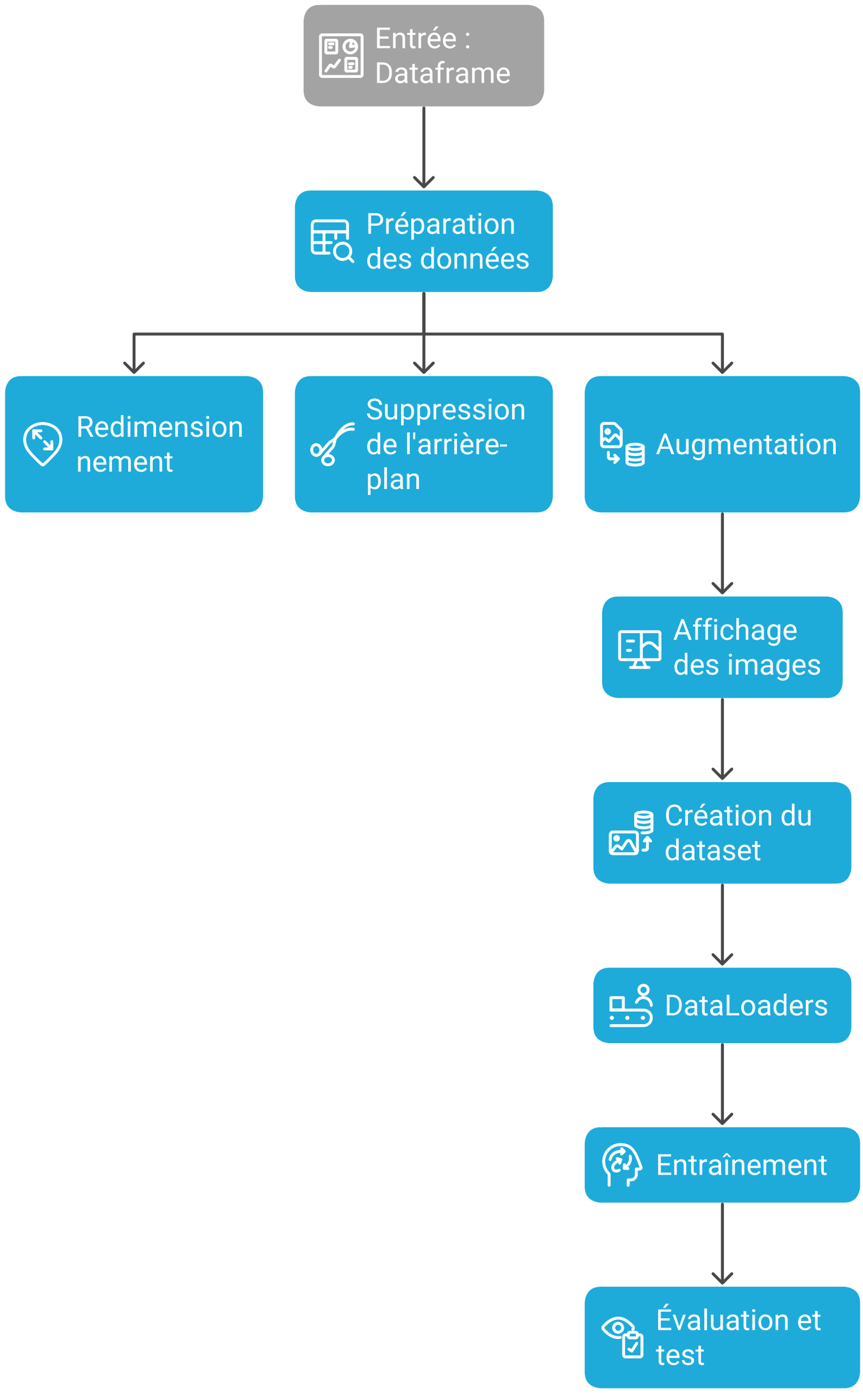
4. **Création du dataset** (**AfricanPlumsFromDataFrame**) :
    - Encode les étiquettes (**Label** et **Defect Type**) avec **LabelEncoder**.
    - Entrée : **augmented_df**
    - Sortie : Dataset PyTorch avec des paires (image, global_label, defect_label)

5. **DataLoaders** :
    - Les données sont divisées en ensembles train/val/test.
    - Les DataLoaders fournissent des lots d'images (format C,H,W après transformation finale à **image_size**) et leurs étiquettes.

6. **Entraînement** :
    - Entrée : Lots d'images (C,224,224), étiquettes globales et de défauts.
    - Sortie : Prédictions, pertes, métriques (précision).

7. **Évaluation et test** :
    - Entrée : Images de test (C,224,224)
    - Sortie : Prédictions, rapports de classification, résultats de test.

Flux des données dans le pipeline de traitement d'images



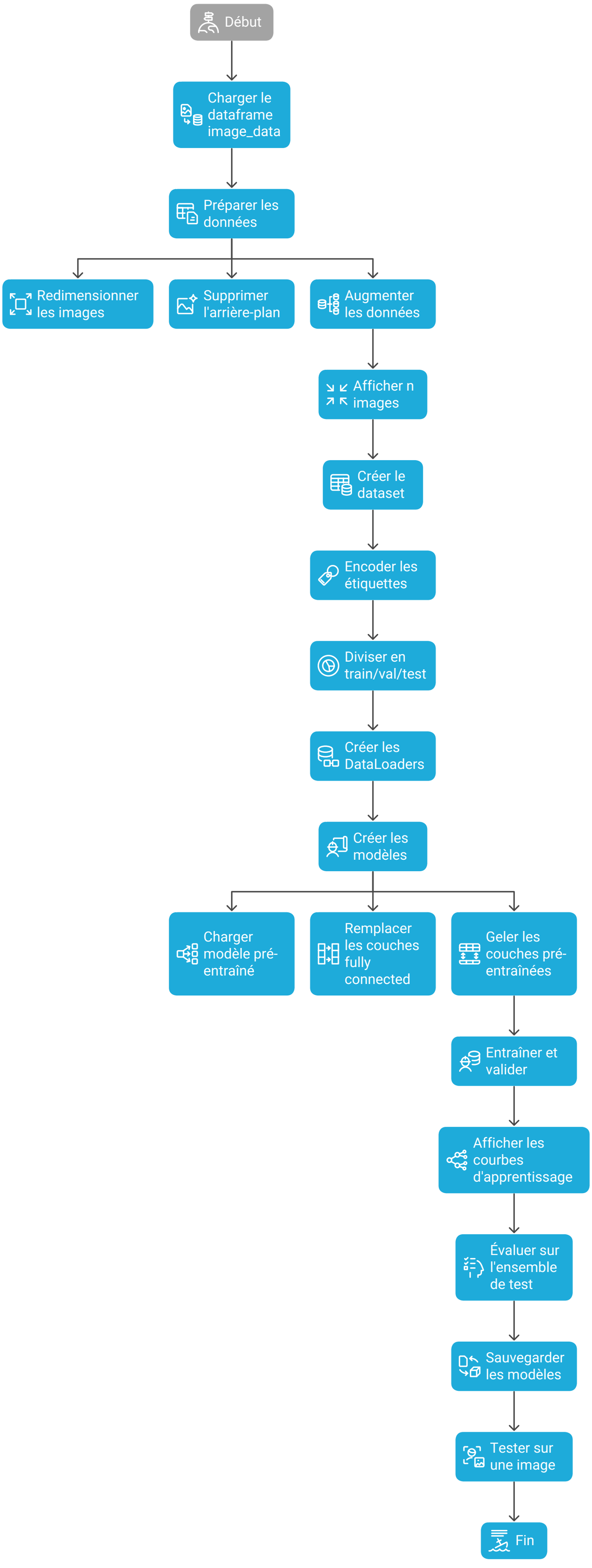
---

3. **Flowchart (Diagramme de flux)**

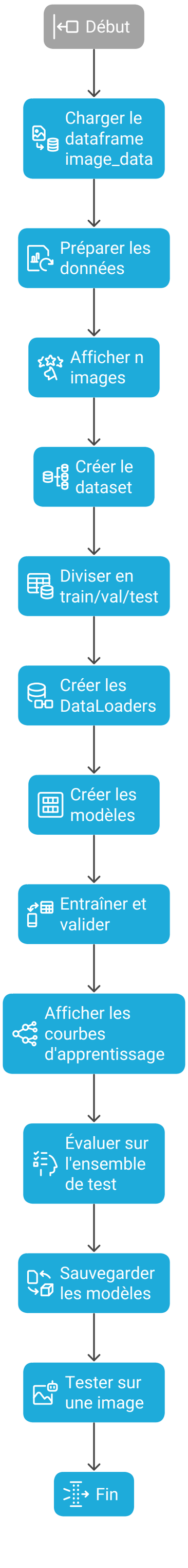
Voici une représentation textuelle du flowchart :

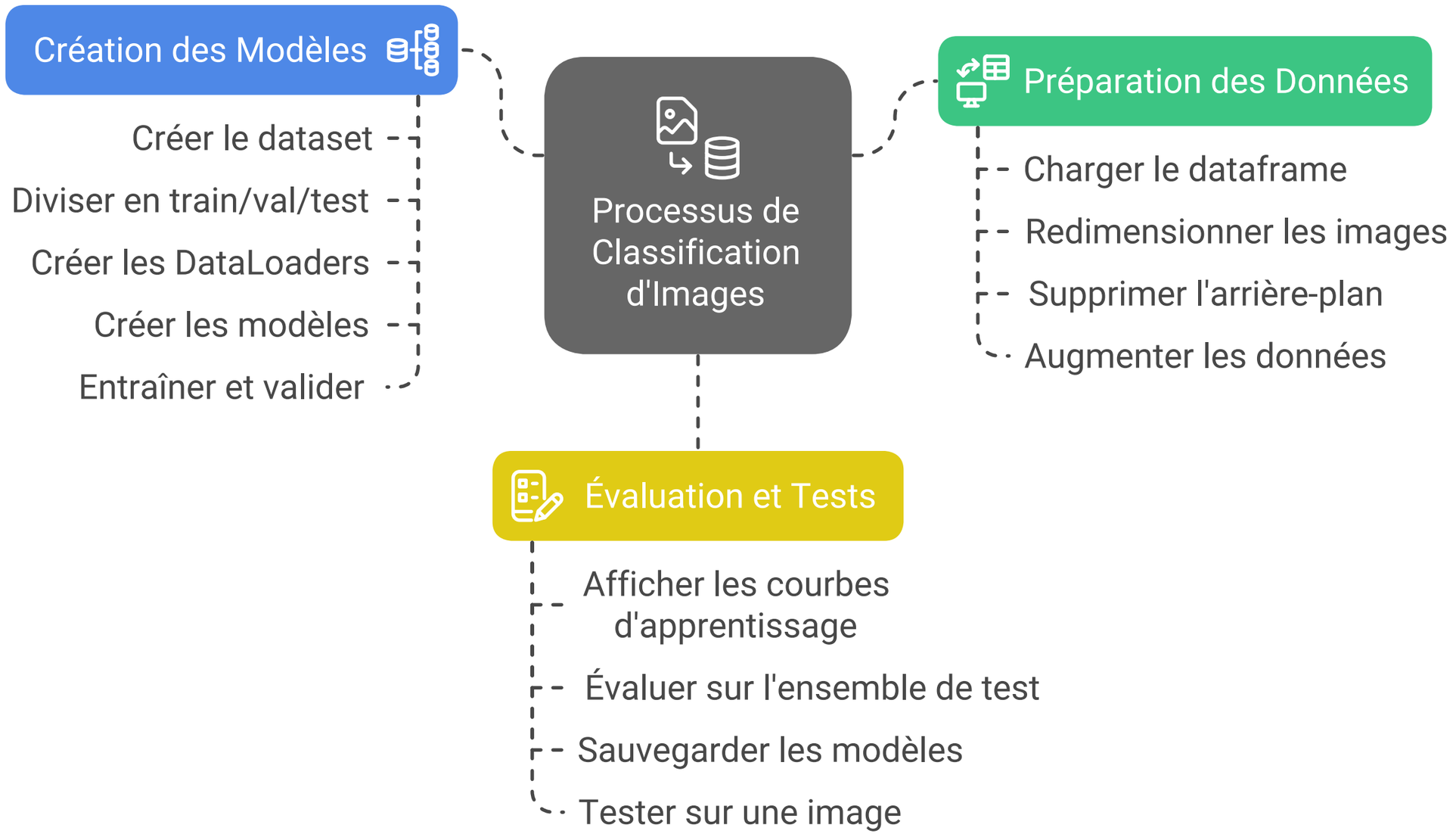
```
[Début]
   |
[Charger le dataframe image_data]
   |
[Préparer les données]
   |—— Redimensionner les images (resize_images: 512x512)
  |—— Supprimer l'arrière-plan (rembg)
  |—— Augmenter les données (Albumentations: équilibrer à 1,721 images par classe)
  |
[Afficher n images (display_n_images)]
  |
[Créer le dataset (AfricanPlumsFromDataFrame)]
  |—— Encoder les étiquettes (LabelEncoder)
  |
[Diviser en train/val/test (70%/15%/15%)]
  |
[Créer les DataLoaders]
  |
[Créer les modèles (create_finetune_models)]
  |—— Charger modèle pré-entraîné (ex: ResNet18)
  |—— Remplacer les couches fully connected (CustomFC)
  |—— Geler les couches pré-entraînées
  |
[Entraîner et valider (train_and_validate)]
  |—— Pour chaque époque :
  |   - Entraîner sur train_loader
  |   - Valider sur val_loader
  |   - Calculer pertes et précisions
  |
[Afficher les courbes d'apprentissage (plot_training_history)]
  |
[Évaluer sur l'ensemble de test (evaluate_model)]
  |—— Générer rapports de classification
  |
```

Processus de Préparation et d'Entraînement de Modèles



Flux de travail de classification d'images de prunes africaines



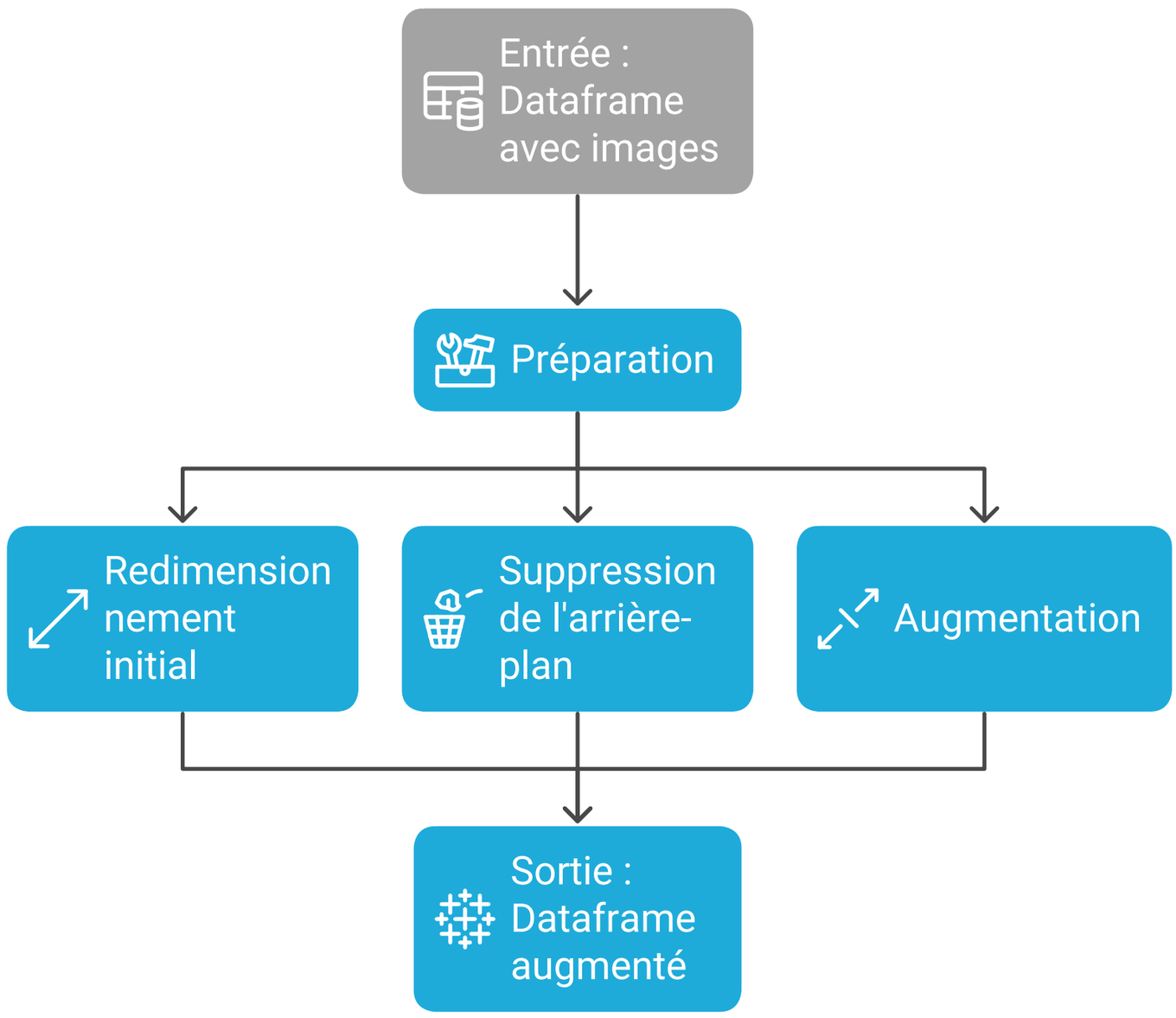


---

4. **Architecture globale**

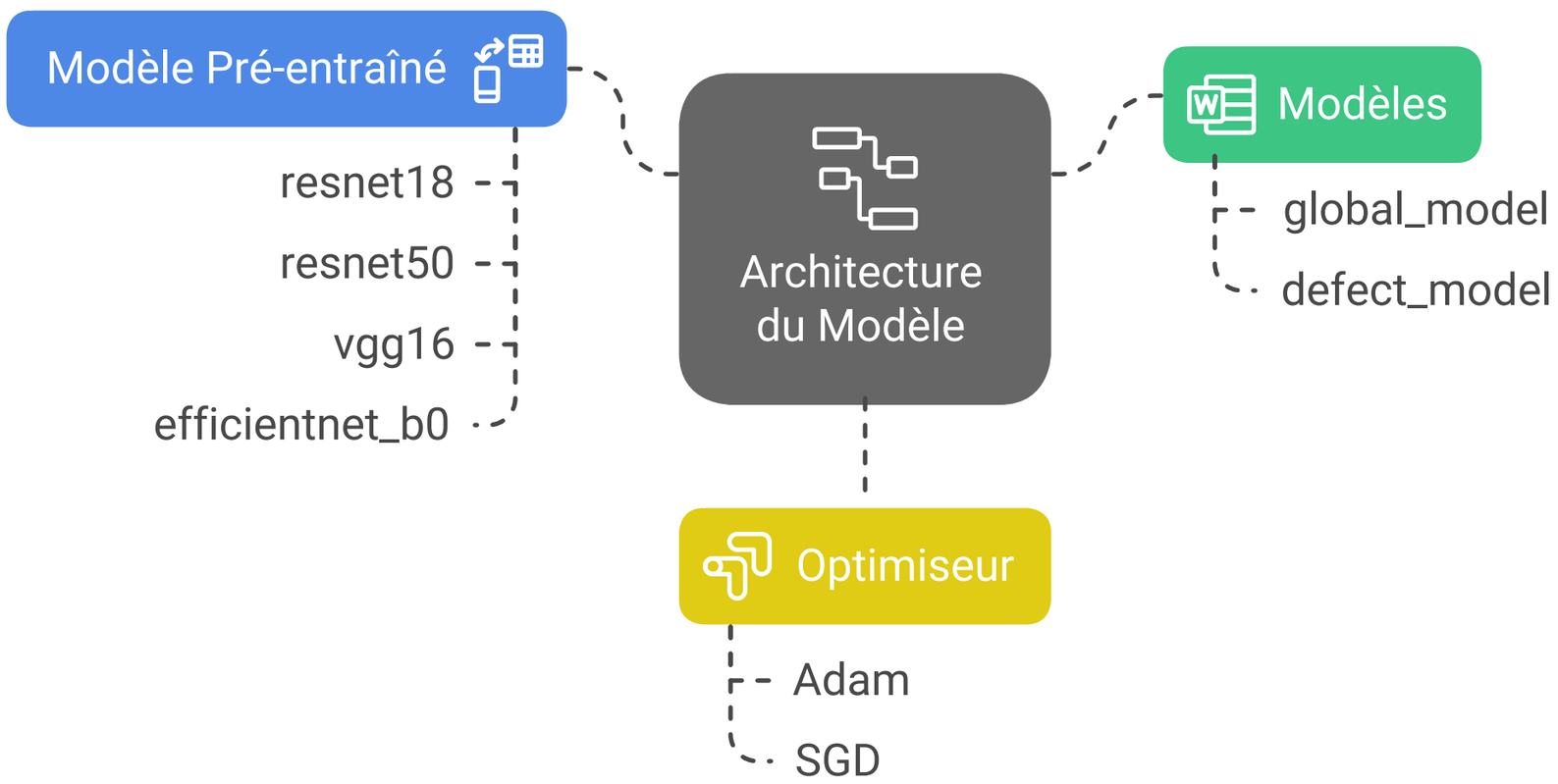
Architecture des données :

- **Entrée** : Dataframe avec images (tableaux NumPy), étiquettes globales (**Label**), et types de défauts (**Defect Type**).
- **Préparation** :
    - Redimensionnement initial (512x512).
    - Suppression de l'arrière-plan (**rembg**).
    - Augmentation (**Albumentations**) pour équilibrer les classes.
- **Sortie** : Dataframe **augmented_df** avec des images prétraitées et augmentées.



Architecture du modèle :

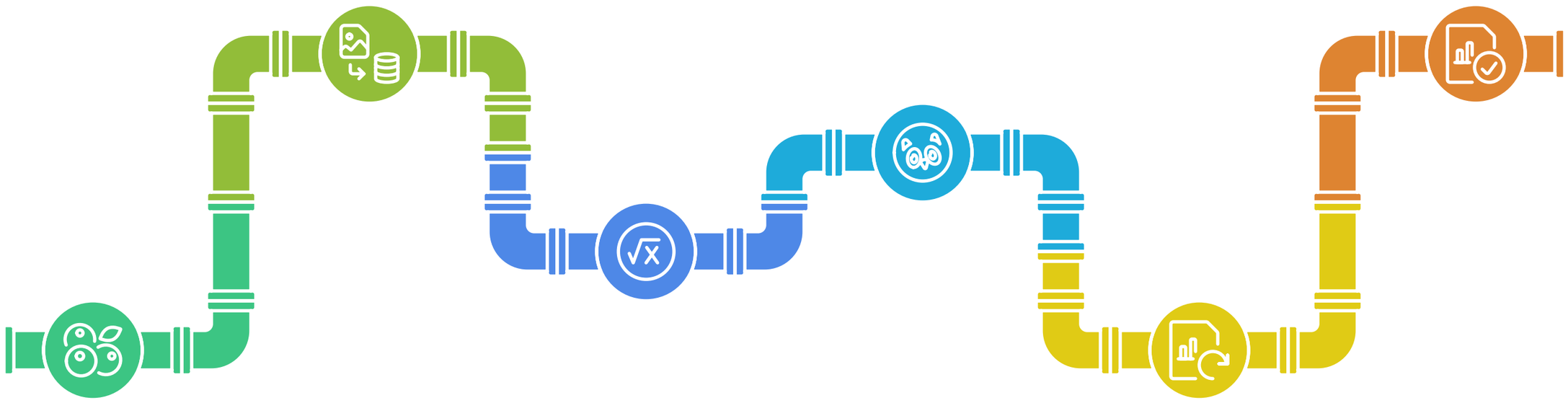
- **Deux modèles** :
    1. **global_model** : Classe les images en **unaffected**, **unripe**, **defective**.
    2. **defect_model** : Classe les images défectueuses en **bruised**, **cracked**, **rotten**, **spotted**.
- **Modèle pré-entraîné** :
    - Choix parmi **resnet18**, **resnet50**, **vgg16**, **efficientnet_b0** (configurable via **HYPERPARAMS['model_name']**).
    - Exemple avec ResNet18 :
        - Couches convolutives pré-entraînées (gelées).
        - Couche fully connected personnalisée (**CustomFC**) :
            - **num_fc_layers** couches (par défaut 2).
            - Taille des couches cachées : **fc_hidden_size** (par défaut 512).
            - Dropout : **dropout_rate** (par défaut 0.5).
- **Optimiseur** : Adam ou SGD (configurable via **HYPERPARAMS['optimizer']**).



Pipeline d'entraînement :

- **Dataset** : **AfricanPlumsFromDataFrame** (images, étiquettes encodées).
- **DataLoaders** : Lots d'images pour entraînement, validation, test.
- **Entraînement** :
    - Perte : CrossEntropyLoss.
    - Optimisation : Adam/SGD.
    - Métriques : Perte, précision (global et défauts).
- **Évaluation** : Rapports de classification (précision, rappel, F1-score).

Processus de Pipeline d'Entraînement pour la Classification d'Images



01

Chargement du Dataset

Charger le dataset d'images de prunes africaines

02

Création de DataLoaders

Préparer des lots de données pour l'entraînement, la validation et le test

03

Définition de la Fonction de Perte

Choisir CrossEntropyLoss pour le calcul de la perte

04

Sélection de l'Algorithme d'Optimisation

Choisir Adam ou SGD pour l'optimisation du modèle

05

Surveillance des Métriques

Suivre la perte et la précision pendant l'entraînement

06

Génération de Rapports d'Évaluation

Produire des rapports de classification avec précision, rappel et F1-score

Sorties :

- Graphiques d'apprentissage ([**training_history.png**]).
- Rapports de classification.
- Modèles sauvegardés.
- Prédictions sur une image de test.

---

5. **Scénario de test en production**

Contexte :
Une application web ou mobile utilise le modèle entraîné pour classer des images de prunes africaines en temps réel. Les utilisateurs (par exemple, agriculteurs ou inspecteurs qualité) téléchargent une image de prune, et l'application retourne la classification globale (**unaffected**, **unripe**, **defective**) et, si défectueuse, le type de défaut (**bruised**, **cracked**, etc.).

Scénario :

1. **Déploiement** :
    - Les modèles entraînés (**global_model_finetuned_resnet18.pth** et **defect_model_finetuned_resnet18.pth**) sont déployés sur un serveur Flask ou FastAPI.
    - Les dépendances (**torch**, **torchvision**, **rembg**, **opencv-python**) sont installées sur le serveur.
    - Le serveur charge les modèles et les met en mode évaluation (**model.eval()**).

2. **Prétraitement de l'image** :

- L'image est convertie en tableau NumPy (format C,H,W).
- L'image est redimensionnée à **resize_height** x **resize_width** (512x512) via **resize_images**.
- L'arrière-plan est supprimé avec **rembg**.
- L'image est redimensionnée à **image_size** (224x224) et normalisée (**transforms.Normalize**).

3. **Prédiction** :

    - L'image prétraitée est passée au **global_model** pour obtenir la classification globale.