Projet de classification d’images d’hématies

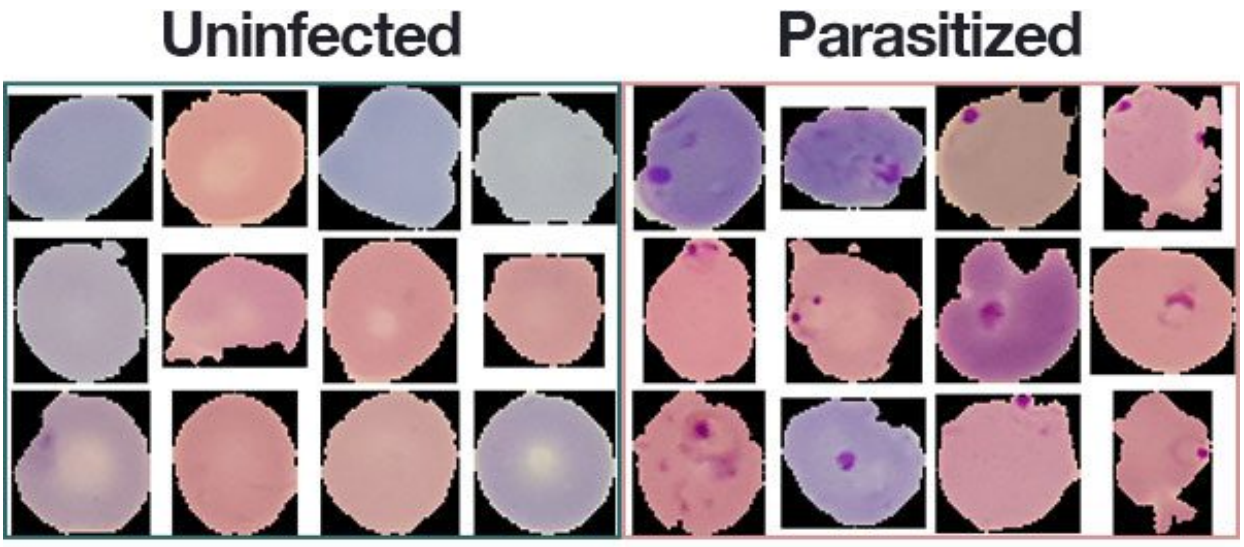
Introduction

Contexte et Importance du Projet

Le paludisme est une maladie infectieuse grave, transmise par les moustiques, qui touche des millions de personnes chaque année, principalement dans les régions tropicales et subtropicales.
La détection précoce et précise de cette maladie est cruciale pour le traitement efficace et la réduction de la mortalité. Cependant, le diagnostic repose souvent sur l'examen microscopique des frottis sanguins, une procédure qui peut être longue et nécessitant une expertise médicale spécialisée.

Objectif du Projet

L'objectif de ce projet est de développer et d'entraîner un modèle de réseau de neurones convolutifs (CNN) capable de différencier les hématies infectées par le paludisme de celles qui sont saines.
Un tel modèle pourrait considérablement accélérer le processus de diagnostic et augmenter sa précision, en offrant une assistance automatisée aux professionnels de santé.



Structure du Projet

Ce projet se déroulera en plusieurs étapes clés :

1. **Manipulation de la Donnée :** Préparation et augmentation des données pour l'entraînement des modèles.
2. **Entraînement de Modèles :** Implémentation et entraînement de trois modèles CNN différents.
3. **Évaluation des Modèles :** Test et évaluation des performances des modèles sur un jeu de données de test.

Attentes et Rendu

- À la fin de ce projet, vous devrez soumettre **un notebook Jupyter complet et bien commenté**, documentant toutes les étapes de la préparation des données à l'évaluation des modèles.
- Vous travaillerez sur ce projet au cours des **FOAD du jeudi 05/02/2026.**
- Vous devrez soumettre votre travail au plus tard le **jeudi 05/02/2024 à 17h**
- **Aucun clone depuis github / Kaggle n'est toléré**
- **Projet à réaliser en binôme ou trinôme**

Remerciement

Cet ensemble de données est extrait du site officiel des NIH :
"https://ceb.nlm.nih.gov/repositories/malaria-datasets/"

I) Manipulation de la Donnée

1) Télécharger les données fournies

- Utilisez le jeu de données fourni dans le répertoire du FOAD. Ces données sont constituées d'images d'hématies parasitées ou saines, munies de leurs labels associés.

2) Charger les données à l'aide de Python

- Utilisez les bibliothèques Python (par exemple, `PIL` ou `opencv` pour les images et `pandas` pour les labels) pour charger les images et leurs labels.

3) Normaliser les images

- Normalisez les images pour que les valeurs des pixels soient comprises entre 0 et 1 en les divisant par 255.
- Faites attention il se peut que les modèles que vous allez fine tuner aient été entraînés avec un jeu de données normalisée différemment (Normalisation standard)

4) Encoder les labels

- Utilisez `LabelEncoder` de `sklearn` pour transformer les labels catégoriels en valeurs numériques.

5) Faire de la data augmentation

- Utilisez `ImageDataGenerator` de Keras pour augmenter les données en appliquant des transformations comme des rotations, des zooms, des flips, etc.

6) Afficher les images augmentées

- Affichez quelques exemples d'images augmentées pour visualiser les transformations appliquées.

II) Entraînement de 3 Modèles

1) Implémentation de 2 Callbacks

- **Early Stopping :** Utilisez `EarlyStopping` de Keras pour surveiller la perte de validation (`val_loss`) et arrêter l'entraînement lorsque celle-ci cesse de s'améliorer.
- **Learning Rate Decay :** Proposez une fonction de décroissance pour le taux d'apprentissage. Par exemple, vous pouvez utiliser `ReduceLROnPlateau` de Keras pour réduire le taux d'apprentissage lorsque la performance cesse de s'améliorer.

2) Implémentation de 3 CNN pour la Labelisation à l'aide de Keras

- **Modèle from scratch avec Sequential** : (ps : n'oubliez pas le dropout)

```
[Input Layer]
      |
      V
[Conv2D: 32 filters, 3x3, ReLU]
      |
      V
[MaxPooling2D: 2x2]
      |
      V
[Conv2D: 64 filters, 3x3, ReLU]
      |
      V
[MaxPooling2D: 2x2]
      |
      V
[Flatten Layer]
      |
      V
[Dense: 128 units, ReLU]
      |
      V
[Output: num_classes, Softmax or Sigmoid]
```

- **Fine-tuning du VGG16 pré-entraîné sur ImageNet** :
    - Récupérez l'encodeur pré-entraîné et ajoutez des couches Dense pour l'adaptation finale.
    - Utilisez la même normalisation dans le pré-traitement des données que celle utilisée lors de l'entraînement initial sur ImageNet.

- **Fine-tuning du ResNet50 pré-entraîné sur ImageNet** :
    - Suivez une démarche similaire à celle du VGG16.

3) Entraînement des 3 modèles à l'aide des Callbacks implémentés

- Utilisez `model.fit` pour entraîner chaque modèle en utilisant les callbacks définis précédemment.

4) Sauvegarder les poids des 3 différents modèles

- Utilisez `model.save_weights` pour sauvegarder les poids des modèles entraînés.

III) Test des Modèles

1) Calculer la Matrice de Confusion

- Utilisez `confusion_matrix` de `sklearn` pour calculer **la matrice de confusion** de chaque modèle sur le jeu de test.

2) Calculer les métriques de performance

- Pour chaque modèle, calculez les métriques suivantes : **accuracy, precision, f1-score, recall, sensibilité et spécificité**. Utilisez les fonctions de `sklearn.metrics`.

3) Afficher le graphique ROC et calculer l'AUC de chacun des 3 modèles.

- Interprétez les résultats obtenus