MLOps

Soutenance Projet

Classification des ordures

Présenté par :

Bintou Tenning NGOM

Fatou MBOUP

Fatou Kiné NDIAYE

Sommaire

Problématique

Problématique

ENJEU ENVIRONNEMENTAL

- Protection de l’Environnement,
- Conservation des Ressources,
- Réduction des Émissions de Gaz à Effet de Serre.

BESOINS DE WASTENET

WasteNet souhaite proposer un objet de tri intelligent afin de recycler certains objets et de diminuer les impacts néfastes sur l'environnement.

COMPLEXITÉ

Mettre en place un modèle prédictif performant pour assurer un tri précis et fiable.

Objectifs

Objectif principal

- **Mettre en place un outil de tri intelligent des ordures**

Objectifs spécifiques

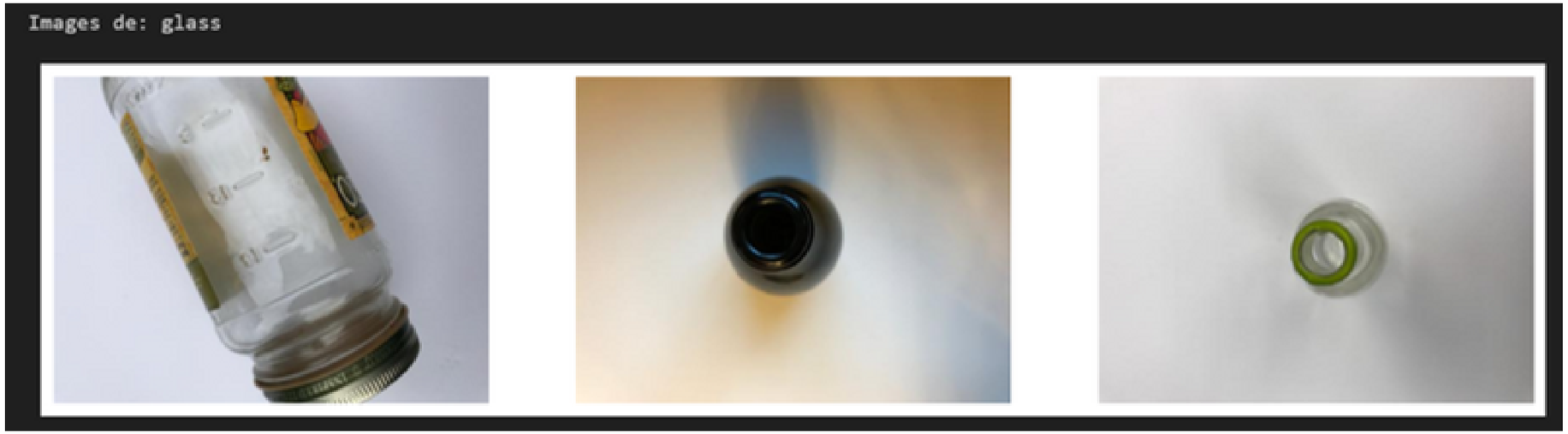
- Mettre en place un modèle puissant pour classer les images d’ordures suivant leurs catégories respectives.
- Amélioration du taux de recyclage
- L’intégration de bonnes pratiques MLOps pour assurer la robustesse et la maintenabilité du système.

Présentation du jeu de données

Données

Le dataset contient **2527** images réparties en **6** classes :

- **Carboard**
- **Paper**
- **Glass**
- **Metal**
- **Plastic**
- **Trash**



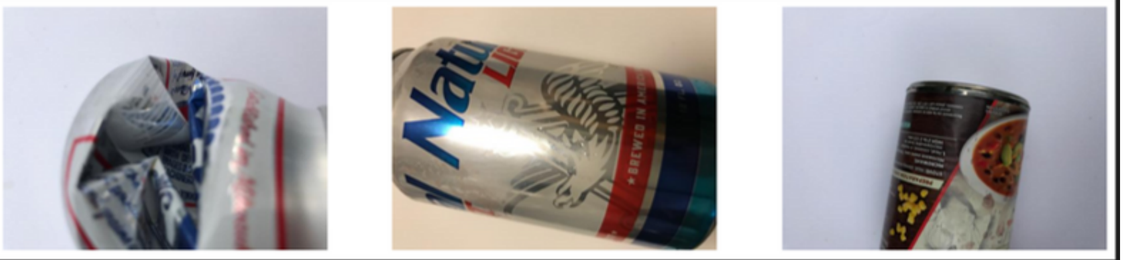
Données

Images de: cardboard

Images de: plastic

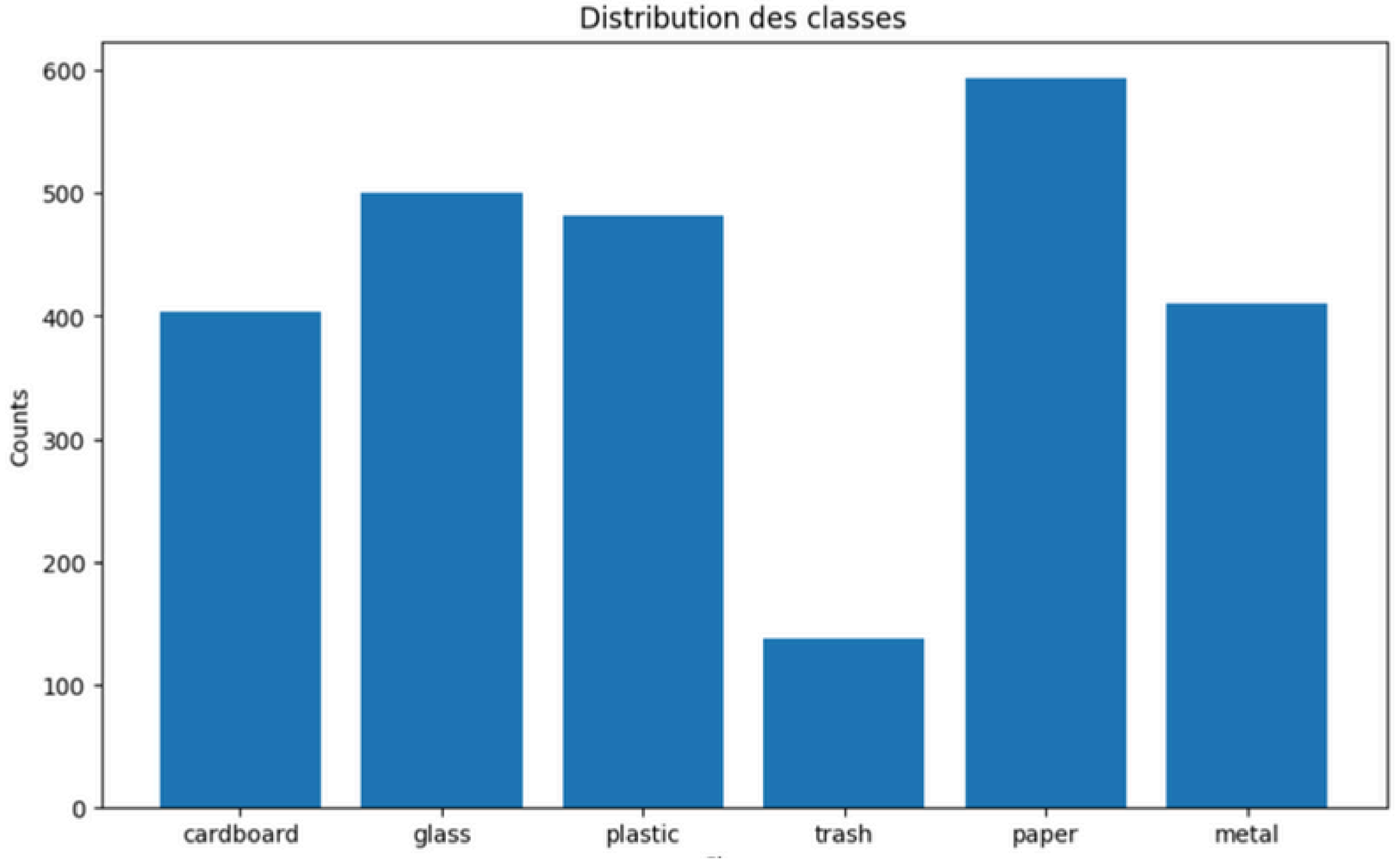
Images de: trash

Images de: metal

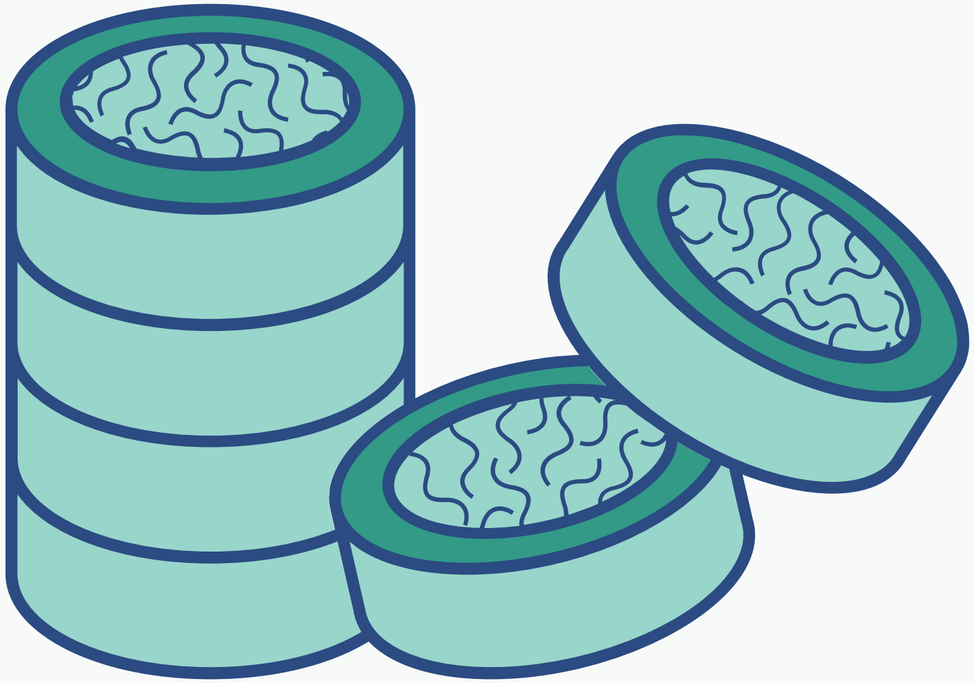


Images de: paper

Distribution des classes



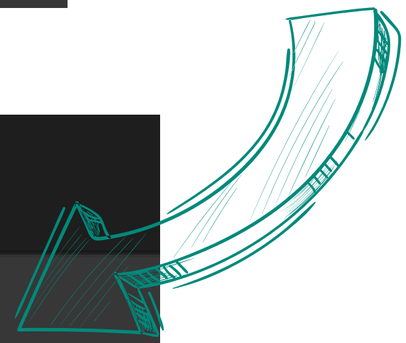
Prétraitement des données



Redimentionnement des images

```
img_dimensions(base_dir)
```

```
Analyse des images en: Garbage_classification/trash
Dimensions les plus courantes:
Dimension (hauteur x largeur): (384, 512), Fréquence: 137
```



```
[ ] processed_dir = "/content/drive/MyDrive/MLFlow_Class_Project/processed_images"
    data, labels = process_dataset(base_dir, processed_dir)
```

```
Classe cardboard: 403 processed
Classe glass: 501 processed
Classe metal: 410 processed
Classe paper: 594 processed
Classe plastic: 482 processed
Classe trash: 137 processed
```

```
img_dimensions(processed_dir)
```

```
Analyse des images en: /content/drive/MyDrive/MLFlow_Class_Project/processed_images/cardbo
Dimensions les plus courantes:
Dimension (hauteur x largeur): (128, 128), Fréquence: 403
```

Normalisation, encodage, augmentation des données

```
# Convertit la liste data en un tableau NumPy.
# Spécifie le type de données comme "float32" pour la précision et l'efficacité
# Divise toutes les valeurs par 255.0, normalisant ainsi les pixels dans la plage [0, 1]
data = np.array(data, dtype="float32") / 255.0
#Convertit la liste labels en un tableau NumPy pour un traitement ultérieur efficace.
labels = np.array(labels)
# Crée une instance de LabelBinarizer de scikit-learn qui va encoder les étiquettes textuell
mlb = LabelBinarizer()
labels = mlb.fit_transform(labels)
#Affiche la première étiquette encodée pour vérification.
print(labels[0])
```

```
[1 0 0 0 0 0]
```

```
[21] final_imgs_data, final_labels_data = increase_dataset(data, labels)
```

```
[22] print("Size before augmentation : ",data.shape[0])
     print("Size After augmentation : ",final_imgs_data.shape[0])
```

```
Size before augmentation :  2527
Size After augmentation :  5054
```

Séparation des données

```
[23] x_train, x_val, x_test, y_train, y_val, y_test = split_data(final_imgs_data, final_labels_data)

[24] # Utilisation de la fonction
     classes = np.arange(labels.shape[1])

     # Vérifiez les distributions des classes dans chaque ensemble
     train_class_distribution = check_class_distribution(y_train, classes)
     val_class_distribution = check_class_distribution(y_val, classes)
     test_class_distribution = check_class_distribution(y_test, classes)

     print("Distribution des classes dans l'ensemble d'entraînement :", train_class_distribution)
     print("Distribution des classes dans l'ensemble de validation :", val_class_distribution)
     print("Distribution des classes dans l'ensemble de test :", test_class_distribution)
```

```
Distribution des classes dans l'ensemble d'entraînement : {0: 468, 1: 589, 2: 491, 3: 716, 4: 606, 5: 162}
Distribution des classes dans l'ensemble de validation : {0: 162, 1: 197, 2: 165, 3: 252, 4: 179, 5: 56}
Distribution des classes dans l'ensemble de test : {0: 176, 1: 216, 2: 164, 3: 220, 4: 179, 5: 56}
```

Modèles et métriques utilisés

Modèles utilisés

DenseNet 121

- Ajout de nouvelles couches:
  - GlobalAveragePooling2D pour réduire la dimensionnalité.
  - BatchNormalization pour la normalisation des activations.
  - Dropout pour la régularisation afin de réduire le surapprentissage.
  - Couches Dense pour la classification.
- augmentation de données lors de l’entrainement

MobileNetV2

- base pré-entraînée
- Ajout de couches personnalisées, une couche de GlobalAveragePooling2D et des couches Dense pour la classification finale
- Callback : Utilisation de early_stopping pour arrêter l'entraînement si la performance se dégrade, afin d'éviter le surapprentissage.

VGG16

- Utilisation de VGG16 avec des poids pré-entraînés sur ImageNet
- Transformation des tableaux numpy en tensors PyTorch et permutation des dimensions pour correspondre au format attendu par VGG16
- Remplacement de la dernière couche de classification (model.classifier[6]) par une couche linéaire adaptée au nombre de classes

Métriques utilisées

- **Accuracy** : mesure la proportion de prédictions correctes parmi toutes les prédictions effectuées.

- **Précision** : mesure la proportion de prédictions positives correctes parmi toutes les prédictions positives faites.

- **Recall** : mesure la proportion de vrais positifs correctement identifiés parmi tous les vrais positifs.

- **Log loss** : mesure la qualité des probabilités prédites. Plus le log loss est bas, meilleure est la performance du modèle.

- **ROC AUC** : mesure la capacité du modèle à distinguer entre les classes positives et négatives. Plus l'AUC est élevée, meilleure est la capacité du modèle à classer les échantillons correctement.

Résultats obtenus

Tracking avec MLFLOW

	Run Name	Created	Models	Metrics: test_accuracy	test_log_loss	test_mean_roc_a	test_precision
☐	garbage_classification_20...	23 minutes ago	DenseNet121 v4	0.88142292...	0.46290358...	0.98406102...	0.88431073...
☐	garbage_classification_20...	26 minutes ago	MobileNetV2 v6	0.52917903...	1.18221701...	0.82343706...	0.62511714...
☐	garbage_classification_20...	42 minutes ago	VGG16 v4	0.53214638...	1.43545602...	0.83290728...	0.65356716...

Parameters (7)

Search parameters

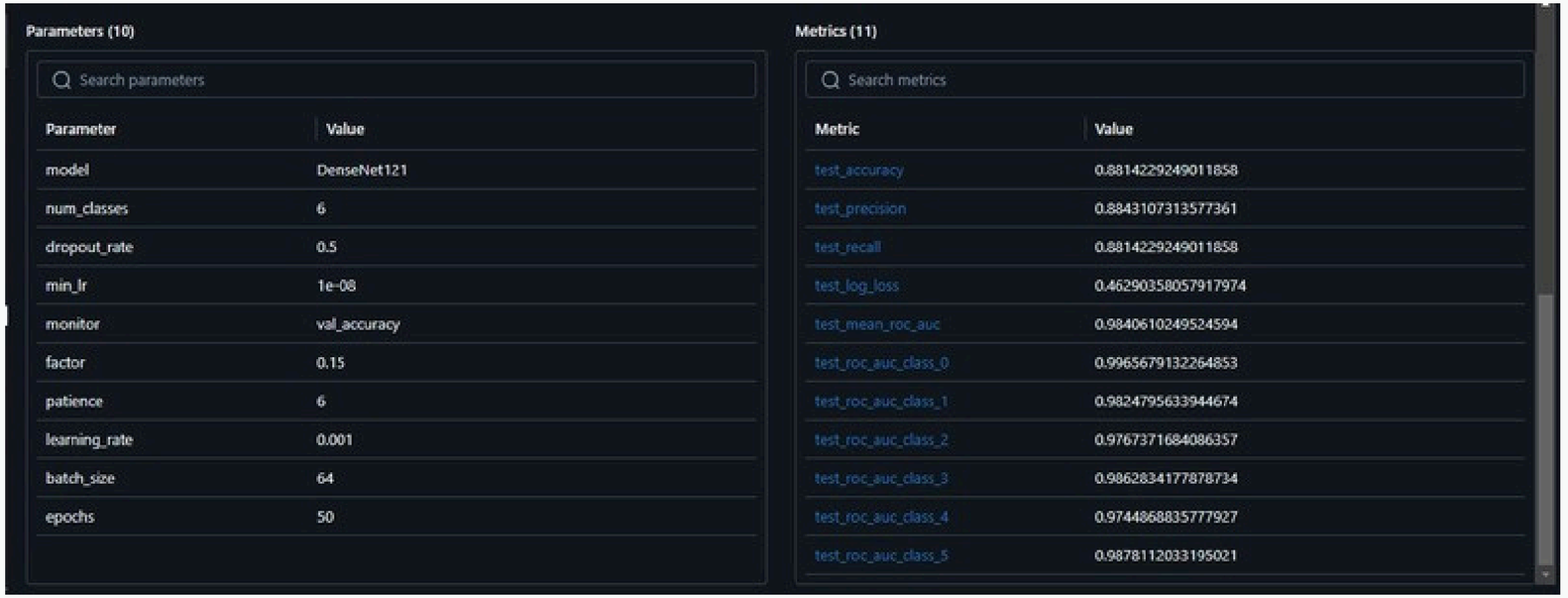
Parameter	Value
model	VGG16
input_shape	(128, 128, 3)
num_classes	6
learning_rate	1e-05
batch_size	32
pretrained	True
epochs	50

Metrics (11)

Search metrics

Metric	Value
test_accuracy	0.5321463897131553
test_precision	0.6535671698090936
test_recall	0.5321463897131553
test_log_loss	1.4354560257857445
test_mean_roc_auc	0.8329072803254877
test_roc_auc_class_0	0.8530450462710941
test_roc_auc_class_1	0.8263248311204285
test_roc_auc_class_2	0.8434575402424627
test_roc_auc_class_3	0.8748046201586024
test_roc_auc_class_4	0.805140739149119
test_roc_auc_class_5	0.7946709050112192

Tracking avec MLFLOW



Parameters (10)

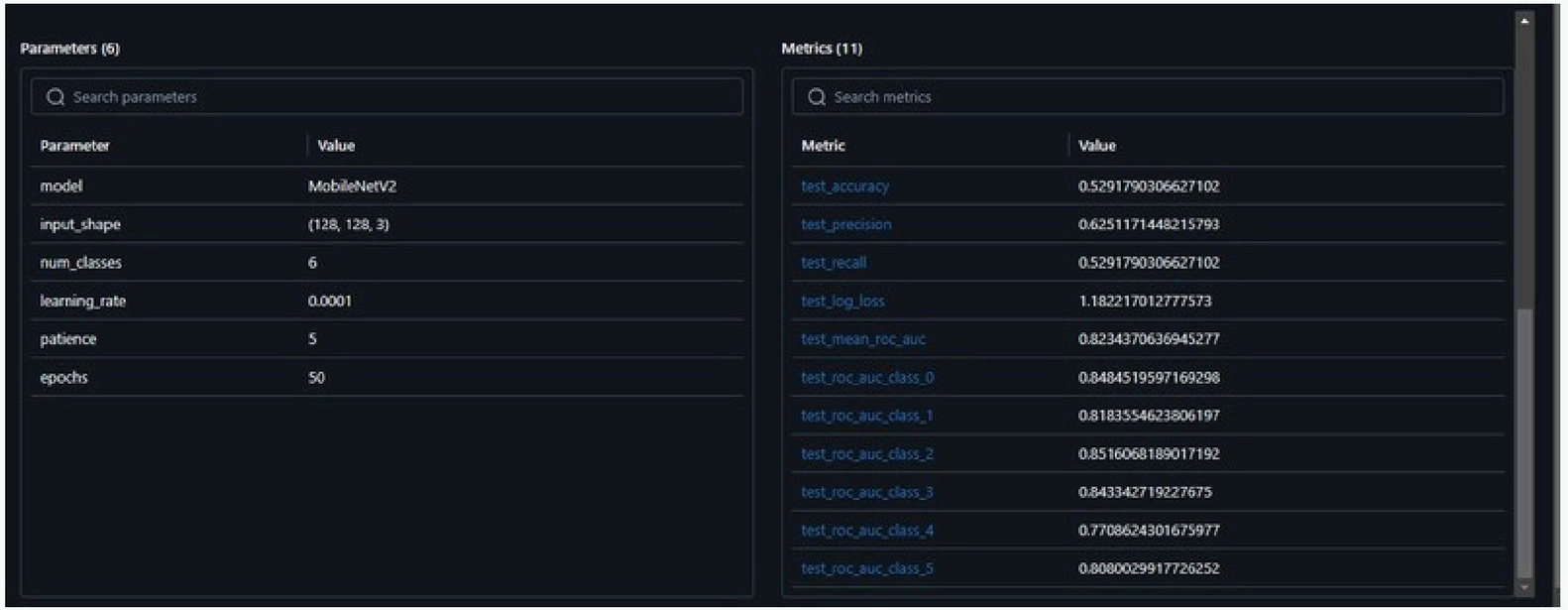
Search parameters

Parameter	Value
model	DenseNet121
num_classes	6
dropout_rate	0.5
min_lr	1e-08
monitor	val_accuracy
factor	0.15
patience	6
learning_rate	0.001
batch_size	64
epochs	50

Metrics (11)

Search metrics

Metric	Value
test_accuracy	0.8814229249011858
test_precision	0.8843107313577361
test_recall	0.8814229249011858
test_log_loss	0.46290358057917974
test_mean_roc_auc	0.9840610249524594
test_roc_auc_class_0	0.9965679132264853
test_roc_auc_class_1	0.9824795633944674
test_roc_auc_class_2	0.9767371684086357
test_roc_auc_class_3	0.9862834177878734
test_roc_auc_class_4	0.9744868835777927
test_roc_auc_class_5	0.9878112033195021



Parameters (6)

Search parameters

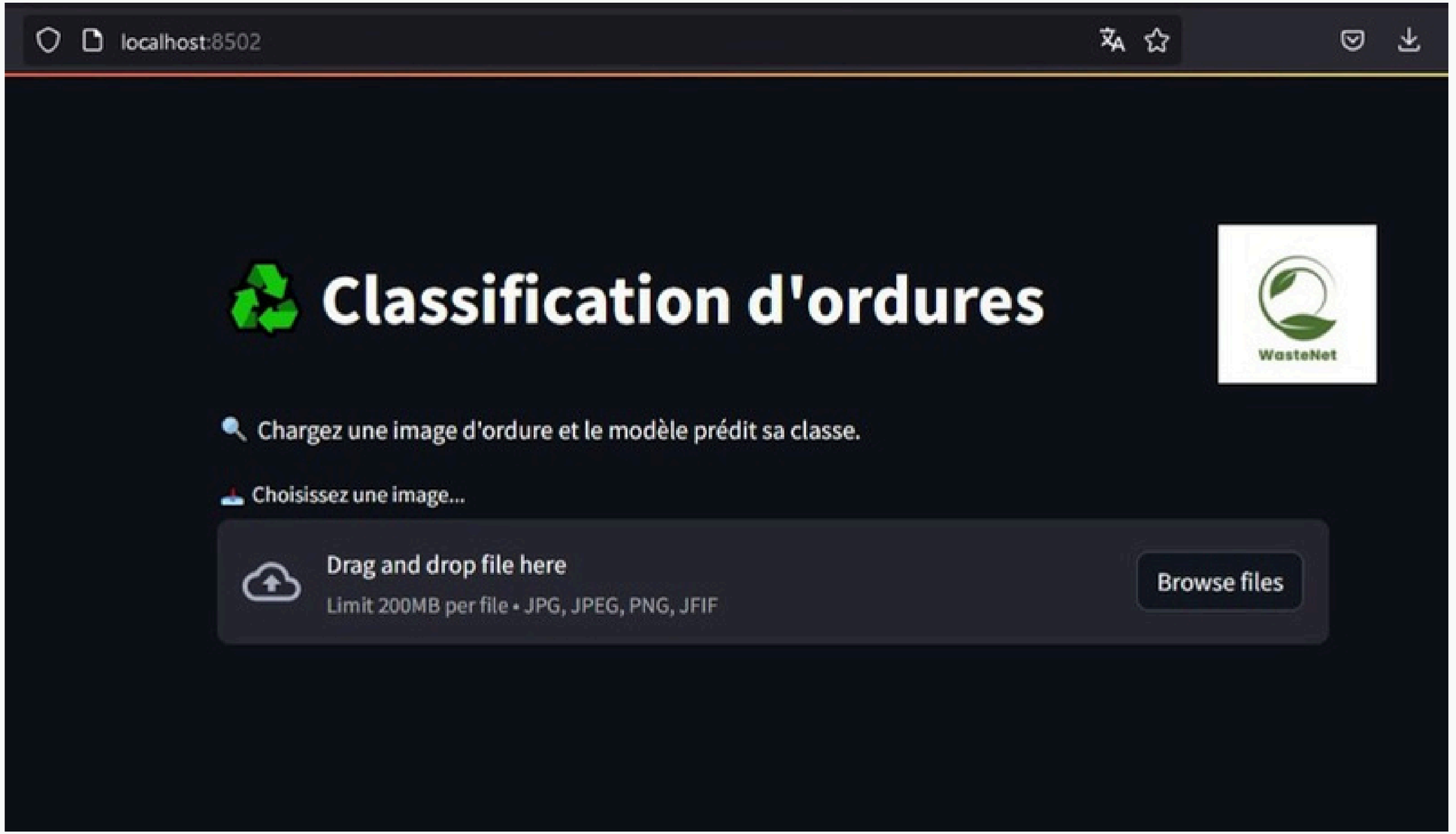
Parameter	Value
model	MobileNetV2
input_shape	(128, 128, 3)
num_classes	6
learning_rate	0.0001
patience	5
epochs	50

Metrics (11)

Search metrics

Metric	Value
test_accuracy	0.5291790306627102
test_precision	0.6251171448215793
test_recall	0.5291790306627102
test_log_loss	1.182217012777573
test_mean_roc_auc	0.8234370636945277
test_roc_auc_class_0	0.8484519597169298
test_roc_auc_class_1	0.8183554623806197
test_roc_auc_class_2	0.8516068189017192
test_roc_auc_class_3	0.843342719227675
test_roc_auc_class_4	0.7708624301675977
test_roc_auc_class_5	0.8080029917726252

Présentation du dashboard


localhost:8502
Classification d'ordures
WasteNet
Chargez une image d'ordure et le modèle prédit sa classe.
Choisissez une image...
Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG, JFIF
Browse files

Sources

- https://www.kaggle.com/datasets/asdasdasasdas/garbage-classification?datasetId=81794&sortBy=voteCount
- https://keras.io/api/applications/vgg/
- https://keras.io/api/applications/mobilenet/
- https://pytorch.org/hub/pytorch_vision_densenet/
- https://mlflow.org/docs/latest/getting-started/index.html
- https://streamlit.io/

Merci de votre aimable attention!

Toute question ou suggestion est la bienvenue !