Détection et comptage d'objets par IA

Recherche Bibliographique : Techniques de détection d'objets

1) État de l'art des techniques de détection d'objets

La détection d'objets est une tâche essentielle dans le domaine de la vision par ordinateur. Elle consiste à localiser et identifier des objets dans une image. Les approches existantes peuvent être classées en deux grandes catégories : les méthodes classiques et celles basées sur le deep learning.

a) Approches classiques

- Caractéristiques locales :
  - **SIFT (Scale-Invariant Feature Transform)** et **SURF (Speeded-Up Robust Features)** permettent d'extraire des points d'intérêt invariants aux transformations d'échelle et de rotation. Ces méthodes sont très efficaces pour des objets avec des textures distinctes.
  - **HOG (Histogram of Oriented Gradients)** est une technique utilisée principalement pour la reconnaissance d'objets comme les piétons. Elle repose sur l'analyse des gradients locaux dans une image.

- Techniques de segmentation :

  La détection par segmentation consiste à isoler les régions d'intérêt à partir de seuils ou de regroupements des pixels. Les méthodes incluent les approches par régions connexes ou les modèles de contour actif.

- Modèles traditionnels :

  Haar Cascades utilisent des filtres pour détecter des objets, comme les visages, en temps réel. Bien que rapide, cette approche est sensible aux variations d'éclairage.

b) Approches basées sur le deep learning

- **R-CNN** et ses variantes :
  - **R-CNN (Regions with Convolutional Neural Networks)** : Cette approche génère des propositions de régions d'intérêt et les classifie avec un réseau convolutif.
  - **Fast R-CNN** et **Faster R-CNN** améliorent la vitesse en intégrant directement les propositions dans le réseau.

- **YOLO (You Only Look Once)** :
  - **YOLO** traite l'image en un seul passage, ce qui permet une détection rapide et en temps réel.
  - Les versions récentes (YOLOv4, YOLOv5) améliorent la précision et la robustesse.

- **SSD (Single Shot MultiBox Detector)**
  - Comme YOLO, SSD est conçu pour une détection rapide. Il détecte des objets à différentes échelles dans une image.

- **Vision Transformers (ViT)**
  - Récemment, les Transformers ont été appliqués à la vision par ordinateur pour capturer des relations globales entre les objets d'une image. Ces modèles montrent des performances prometteuses pour la détection d'objets complexes.

2) Comparaison des approches classiques vs deep learning

Critère	Approches Classiques	Deep Learning
Précision	Modérée, dépend des caractéristiques	Haute précision avec des modèles complexes
Robustesse	Sensibles aux variations (lumière, angle)	Robustesse à des conditions variées
Exigences en données	Peu exigeantes	Nécessitent de grandes bases d'images
Exigences en calcul	Faibles	Elevées (GPU nécessaire)
Facilité d'implémentation	Simples	Complexes

3) Avantages et limitations des méthodes

Méthodes	Avantages	Limitations
Classiques	Simples à mettre en œuvre.	Peu robustes face aux variations.
	Adaptées aux systèmes embarqués et aux environnements avec peu de données.	Moins adaptées à des objets complexes ou variés.
Deep learning	Performances élevées sur des ensembles de données diversifiés.	Exigent des ressources matérielles importantes (GPU).
	Capacité à généraliser pour des cas non vus.	Difficultés pour traiter des jeux de données de petite taille.

Critères et Choix des Techniques Utilisées

1) Pytorch vs TensorFlow

PyTorch

PyTorch est un framework open-source de deep learning, développé par Meta AI et publié en 2017. Conçu pour être convivial et intégré de manière native à Python, il se distingue par :

- Sa gestion dynamique des graphes computationnels, permettant des modifications rapides et intuitives.
- Sa flexibilité pour le prototypage et la recherche.
- Son efficacité en termes d'utilisation mémoire.

PyTorch est particulièrement adapté pour des projets de recherche et des tâches complexes comme la vision par ordinateur et le traitement du langage naturel. Bien qu'il ne soit pas optimisé nativement pour la production, des outils comme Flask ou Django permettent de déployer des modèles entraînés.

TensorFlow

TensorFlow, lancé par Google en 2015, est un framework de deep learning mature et robuste. Il propose des fonctionnalités avancées, notamment :

- Des outils intégrés pour la visualisation (TensorBoard).
- Une capacité de déploiement en production prête à l'emploi grâce à TensorFlow Serving.
- Une compatibilité étendue avec diverses plateformes, y compris mobiles.

Ce framework utilise des graphes computationnels statiques, ce qui peut rendre la prise en main plus complexe, mais il excelle dans les environnements de production nécessitant des solutions évolutives.

Conclusion

Bien que TensorFlow soit un choix privilégié pour les systèmes de production à grande échelle, **nous avons choisi PyTorch** pour notre projet en raison de sa simplicité, de sa flexibilité et de son adéquation aux petits projets de recherche comme le nôtre.

2) HOG vs SIFT

HOG (Histogram of Oriented Gradients)

HOG est une méthode de description d'images qui analyse la forme et la structure des objets en calculant un histogramme d'orientations des gradients dans des zones localisées d'une image

- **Étapes principales :**
  1) Conversion de l'image en niveaux de gris.
  2) Découpage en cellules (par exemple, 8x8 pixels).
  3) Calcul des gradients pour chaque cellule.
  4) Normalisation des blocs (groupes de cellules).
  5) Concaténation des descripteurs pour obtenir une représentation globale.

HOG est particulièrement efficace pour la détection d'objets rigides comme les silhouettes humaines, notamment en atténuant l'influence des ombres et des variations d'illumination.

SIFT (Scale-Invariant Feature Transform)

SIFT est une méthode d'extraction de points-clés dans les images, conçue pour être invariante aux échelles, rotations, et changements d'éclairage.

- **Étapes principales** :
  1. Construction d'un espace échelle via un Gaussian Pyramid pour détecter les points-clés à différentes échelles.
  2. Localisation des points-clés en éliminant ceux ayant une faible stabilité ou contraste.
  3. Attribution d'une orientation aux points-clés.
  4. Génération de descripteurs en analysant les pixels environnants, créant des vecteurs robustes pour la correspondance.

SIFT est souvent utilisé pour des environnements complexes ou pour des objets présentant des déformations non rigides.

Différences principales entre HOG et SIFT

1) Type d'objets détectés :
   - HOG : Idéal pour les objets rigides (silhouettes humaines, voitures).
   - SIFT : Meilleur pour les objets dans des environnements variés ou avec des transformations complexes.
2) Résolution spatiale et invariance :

- HOG : Conserve une certaine résolution spatiale et est insensible aux variations locales de contraste.
- SIFT : Invariant aux échelles, rotations et variations d'éclairage, mais moins performant pour les objets rigides.

3) Performance :
   - HOG est plus rapide et efficace pour des tâches spécifiques comme la détection de piétons.
   - SIFT offre une précision supérieure dans des tâches complexes mais est plus coûteux en termes de calcul.

Conclusion

HOG est le choix optimal grâce à sa simplicité, son efficacité pour capturer les contours rigides des lettres et sa compatibilité avec des modèles comme SVM ou des approches probabilistes simples.

3) Machine Learning vs Deep Learning

Machine Learning

Le **Machine Learning (ML)** est une branche de l'intelligence artificielle où les algorithmes apprennent à partir de données structurées et quantitatives. Dans un cas pratique, comme la prédiction d'un achat, le Data Scientist sélectionne manuellement les variables pertinentes, comme l'âge, le sexe ou le revenu. Cette étape, appelée **feature extraction**, permet de réduire la complexité du problème en ne conservant que les informations les plus pertinentes pour entraîner le modèle.

Deep Learning

Le **Deep Learning (DL)** est une sous-catégorie du Machine Learning qui exploite des réseaux de neurones artificiels inspirés du fonctionnement du cerveau humain. Contrairement au ML, le Deep Learning traite des données non-structurées comme des images, du son ou du texte, sans nécessiter d'étape de feature extraction. Par exemple, pour reconnaître un chien ou un chat sur une image, l'algorithme apprend automatiquement quelles caractéristiques des pixels sont déterminantes.

Conclusion

Dans notre cas, en raison de la **limitation de la taille et de la structure de la base de données**, le Machine Learning s'est avéré être la meilleure approche. Sa capacité à travailler efficacement sur des données structurées avec une intervention humaine pour sélectionner les variables pertinentes en fait un choix idéal pour résoudre notre problème.

Description de l'implémentation

1) BayesianClassifier (Classificateur Bayésien)

Le **BayesianClassifier** est un classificateur probabiliste basé sur le théorème de Bayes, spécifiquement conçu pour classer des objets dans une image en fonction de leurs caractéristiques extraites. Ce classificateur utilise les caractéristiques HOG (Histogram of Oriented Gradients) des objets présents dans les images.

Attributs principaux :

- `feature_means` : Dictionnaire contenant la moyenne des caractéristiques pour chaque classe.
- `feature_variances` : Dictionnaire contenant la variance des caractéristiques pour chaque classe.
- `class_priors` : Probabilité a priori de chaque classe (fréquence relative d'apparition des classes).
- `classes` : Liste des classes identifiées lors de l'entraînement.

Méthodes principales :

- **`extract_features(image)`** :
  - Cette méthode extrait les caractéristiques HOG des objets dans l'image. L'image est d'abord convertie en niveaux de gris (si elle est en couleur), puis une image binaire est obtenue via un seuillage adaptatif pour mieux gérer les variations de luminosité.
  - Ensuite, les contours des objets sont détectés, et pour chaque contour suffisamment grand, un sous-ensemble de l'image est extrait, redimensionné à une taille fixe (28x28 pixels), et les caractéristiques HOG sont calculées.
  - Les caractéristiques sont ensuite normalisées.
- **`train(catalog_path)`** :
  - Lors de l'entraînement, la méthode parcourt un répertoire contenant des images réparties dans des sous-dossiers correspondant à différentes classes. Pour chaque image, les caractéristiques sont extraites et stockées.
  - Ensuite, la méthode calcule la moyenne, la variance et la probabilité a priori pour chaque classe en fonction des images collectées.
- **`predict(image)`** :
  - Cette méthode prédit la classe d'une image donnée en calculant la vraisemblance de chaque classe en fonction des caractéristiques extraites.

    - Elle prend également en compte les différentes rotations possibles de l'image, ajustant les poids des rotations pour affiner la prédiction.

- `save_model(model_path)` :
    - Permet de sauvegarder le modèle entraîné sous forme d'un fichier en utilisant PyTorch, afin qu'il puisse être rechargé plus tard.
- `load_model(model_path)` :
    - Charge un modèle préexistant depuis un fichier, restaurent les moyennes, variances et priorités calculées lors de l'entraînement.
- `visualize_model()` :
    - Permet de visualiser les moyennes des caractéristiques extraites pour chaque classe sous forme de graphiques, ce qui peut aider à mieux comprendre la distribution des caractéristiques pour chaque catégorie.

Résumé de l'implémentation :

Le **classifieur bayésien** dans ce code repose sur une approche probabiliste où les caractéristiques HOG sont extraites de l'image, puis sont utilisées pour estimer la probabilité de chaque classe. Les classes sont déterminées à partir de leurs caractéristiques moyennes et de leurs variances, et le classifieur calcule des "postériors" pour chaque classe en fonction des nouvelles observations (images test). La rotation des images est également prise en compte lors de la prediction pour améliorer la précision de la prediction.

Le classifieur utilise également des techniques de gestion des données telles que l'**adaptative thresholding** pour traiter les variations de lumière et **l'ignoration des petits contours** (bruit) pour concentrer l'attention sur les objets significatifs.

Schéma de fonctionnement : **Image illustrant le fonctionnement de l'entrainement du classifieur Bayesien**

2) KMeansClassifier (Classificateur KMeans)

Le **KMeansClassifier** est un classificateur basé sur l'algorithme de clustering K-means. Contrairement au classifieur bayésien, qui est probabiliste, le K-means est une méthode de clustering non supervisée qui regroupe les données en **k clusters** (groupes) sur la base de la proximité des caractéristiques extraites des images.

Attributs principaux :

- `num_clusters` : Le nombre de clusters (groupes) à générer lors de l'entraînement, par défaut 3.
- `centroids` : Les centroïdes des clusters après l'entraînement.
- `classes` : Liste des classes identifiées lors de l'entraînement.

Méthodes principales :

- **`extract_features(image)`** :
  - Cette méthode fonctionne de la même manière que dans le `BayesianClassifier`. Elle extrait les caractéristiques HOG de chaque image après l'avoir convertie en niveaux de gris, puis l'application d'un seuillage adaptatif et la détection des contours.
  - Les caractéristiques extraites sont ensuite normalisées pour éviter les biais dus aux tailles des contours.
- **`initialize_centroids(features)`** :
  - Cette méthode initialise les centroïdes de manière aléatoire à partir des points de données (caractéristiques extraites), en sélectionnant `num_clusters` points aléatoires.
- **`assign_clusters(features, centroids)`** :
  - Chaque point de donnée (caractéristique extraite) est assigné au cluster dont le centroïde est le plus proche, en calculant les distances euclidiennes entre les points et les centroïdes.
- **`update_centroids(features, cluster_assignments)`** :
  - Après l'assignation des points aux clusters, cette méthode met à jour les centroïdes en calculant la moyenne des points assignés à chaque cluster.
- **`train(catalog_path, num_iterations=100)`** :
  - Cette méthode entraîne le modèle KMeans en parcourant les images d'un répertoire donné. Pour chaque image, elle extrait les caractéristiques, puis elle applique l'algorithme K-means pour déterminer les centroïdes et affecter les points aux clusters.
  - Le processus est répété pendant un certain nombre d'itérations (`num_iterations`), permettant aux centroïdes de converger.
- **`predict(image)`** :

    - Cette méthode prédit la classe d'une image en extrayant ses caractéristiques et en l'assignant au cluster le plus proche. Ensuite, elle retourne la classe associée à ce cluster.

- **`save_model(model_path)`** :
    - Sauvegarde les paramètres du modèle (les centroïdes et les classes) dans un fichier en utilisant PyTorch pour une utilisation future.
- **`load_model(model_path)`** :
    - Charge un modèle préexistant depuis un fichier, en rétablissant les centroïdes et les classes associées.
- **`visualize_model()`** :
    - Visualise les centroïdes des clusters dans un espace 2D. Cette méthode effectue une réduction de dimensionnalité des centroïdes via l'**Analyse en Composantes Principales (PCA)**, ce qui permet de projeter les centroïdes dans un plan 2D. Ensuite, elle affiche les centroïdes avec les noms des classes sur un graphique, ce qui permet de visualiser les groupes et leur relation dans l'espace des caractéristiques.

Résumé de l'implémentation :

Le **classifieur KMeans** utilise un algorithme non supervisé pour regrouper les images en différents clusters, chaque cluster représentant une catégorie (classe) d'objets. L'idée centrale est de réduire l'espace des données (caractéristiques extraites des images) en un nombre de groupes définis par `num_clusters`, et de prédire la classe d'une image en fonction du cluster auquel elle appartient. Contrairement au classifieur bayésien, qui utilise des probabilités basées sur des modèles statistiques, le K-means se base sur la proximité des caractéristiques dans un espace multidimensionnel.

Schéma de fonctionnement : **Image illustrant le fonctionnement de l'entrainement du classifieur Kmeans**

3) ObjectDetectionPipeline (Pipeline de détection d'objets)

L'**ObjectDetectionPipeline** est une classe dédiée à la détection et à la classification d'objets dans des images, en utilisant un modèle de classification personnalisé (par exemple, un classificateur bayésien ou un classificateur kmeans). Ce pipeline effectue une série d'étapes allant du prétraitement des images à la classification, en passant par la détection de contours et l'annotation des résultats.

Attributs principaux :

- **`image_path`** : Le chemin de l'image à traiter.
- **`image`** : L'image chargée depuis le disque, utilisée pour l'inférence et la visualisation.
- **`model`** : Le modèle de classification utilisé pour prédire les objets détectés dans l'image.

Méthodes principales :

- **`load_model(model_path: str, instance_classifier: Classifier=None)`** :
  - Charge un modèle préexistant (par exemple un classificateur bayésien ou le kmeans).
  - Si le modèle existe à l'emplacement spécifié, il est chargé et prêt à être utilisé pour la détection et la classification des objets.
- **`load_image()`** :
  - Charge l'image à partir du chemin spécifié par **image_path**.
  - Si l'image n'est pas trouvée, une exception est levée.
- **`preprocess_image()`** :
  - Divise l'image en 3 images (R, G, B) puis chaque image est binarisé puis refusionné en une seule binarisé.
- **`detect_and_classify_objects()`** :
  - Effectue la détection des objets dans l'image en utilisant l'algorithme de détection de contours de OpenCV.
  - Chaque contour détecté est ensuite extrait, prétraité et classifié en utilisant le modèle chargé (par exemple, classificateur bayésien ou classifieur kmeans).
  - Retourne un dictionnaire des classes détectées et le nombre de leurs occurrences, ainsi qu'une liste des objets détectés avec leurs coordonnées dans l'image.
- **`display_results()`** :

    - Affiche les résultats de la détection et de la classification dans l'image.
    - Cette méthode affiche l'image annotée avec des rectangles et des textes représentant les objets détectés, ainsi qu'un graphique des classes et de leurs fréquences.
- **`display_binary_image()`:**
    - Affiche l'image de départ binarisé (0 ou 255) (noir / blanc)
- **`display_image_with_classes()`** :
    - Affiche l'image avec des classes prédites, où chaque objet détecté est étiqueté avec la classe prédite.
- **`display_image_with_annotations()`** :
    - Affiche l'image avec des annotations, y compris des rectangles autour des objets détectés et les étiquettes correspondantes.
- **`display_classes_count()`** :
    - Affiche un graphique montrant le nombre d'objets détectés pour chaque classe.

Résumé de l'implémentation :

Le **ObjectDetectionPipeline** est un pipeline conçu pour détecter et classer des objets dans une image. Il commence par charger l'image et le modèle de classification, puis effectue un prétraitement de l'image pour en extraire les caractéristiques pertinentes (comme les contours). Ces caractéristiques sont ensuite utilisées par le modèle de classification pour prédire la classe de chaque objet détecté. Le pipeline fournit plusieurs fonctionnalités pour afficher les résultats, telles que l'affichage d'annotations sur l'image et la visualisation des fréquences des classes détectées.

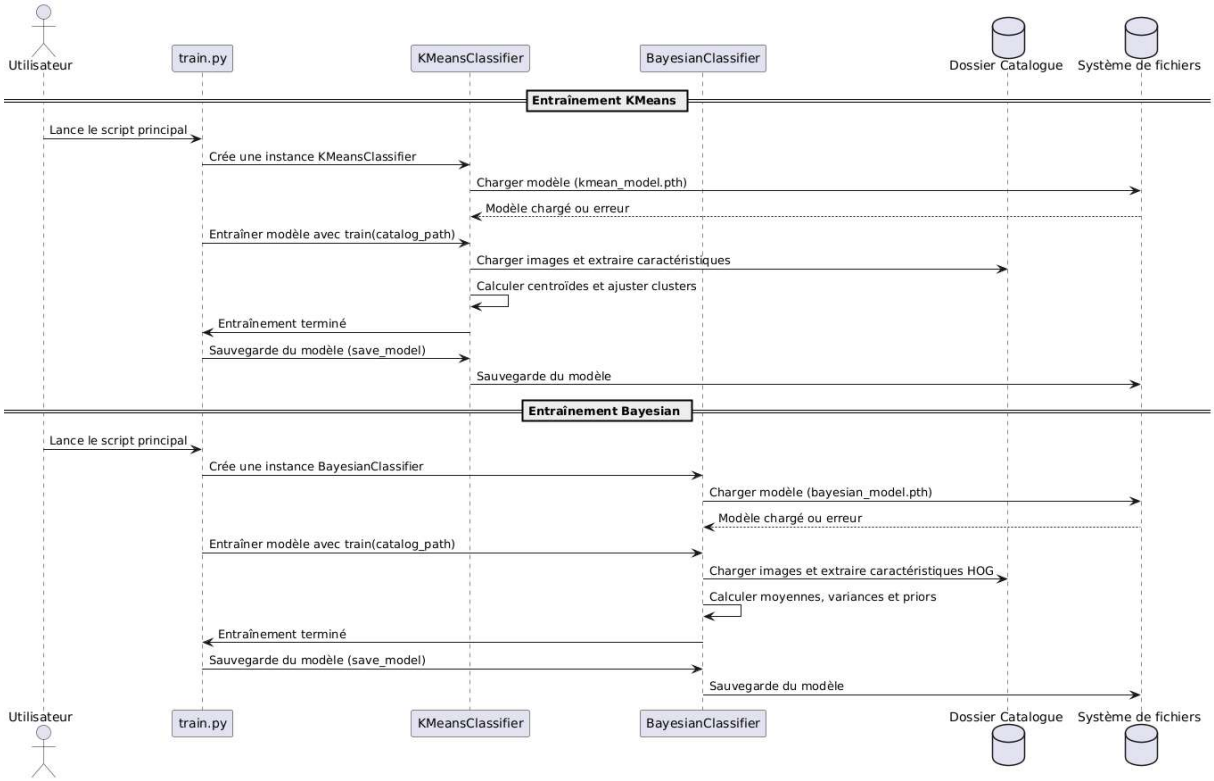
Schéma de fonctionnement : **Image illustrant le fonctionnement de la pipeline**

4) Fonctionnement de l'entrainement des modèles (train.py)

Les classificateurs KMeansClassifier et BayesianClassifier héritent de la classe abstraite Classifier, ce qui permet d'uniformiser l'entraînement et la prédiction dans la pipeline.
Chaque modèle peut être chargé depuis un fichier existant ou entraîné à partir de zéro.
Les images sont traitées pour extraire des caractéristiques :

- **KMeans** ajuste les centroïdes et clusters.
- **Bayesian** calcule moyennes, variances et probabilités a priori à partir des caractéristiques HOG.

Les modèles entraînés sont ensuite sauvegardés pour un usage ultérieur.



5) Fonctionnement de la détection des classes dans l'image (main.py)

Ce diagramme explique le processus de détection et de classification des objets dans une image fournie par l'utilisateur.

L'utilisateur fournit **deux entrées** :

- Une **image** à analyser
- Un **modèle de classification** pré-entrainé

L'utilisateur instancie le pipeline de détection et charge un modèle de classification pré-entrainé.

L'image est ensuite chargée et prétraitée à l'aide de techniques de binarisation et de détection de contours.

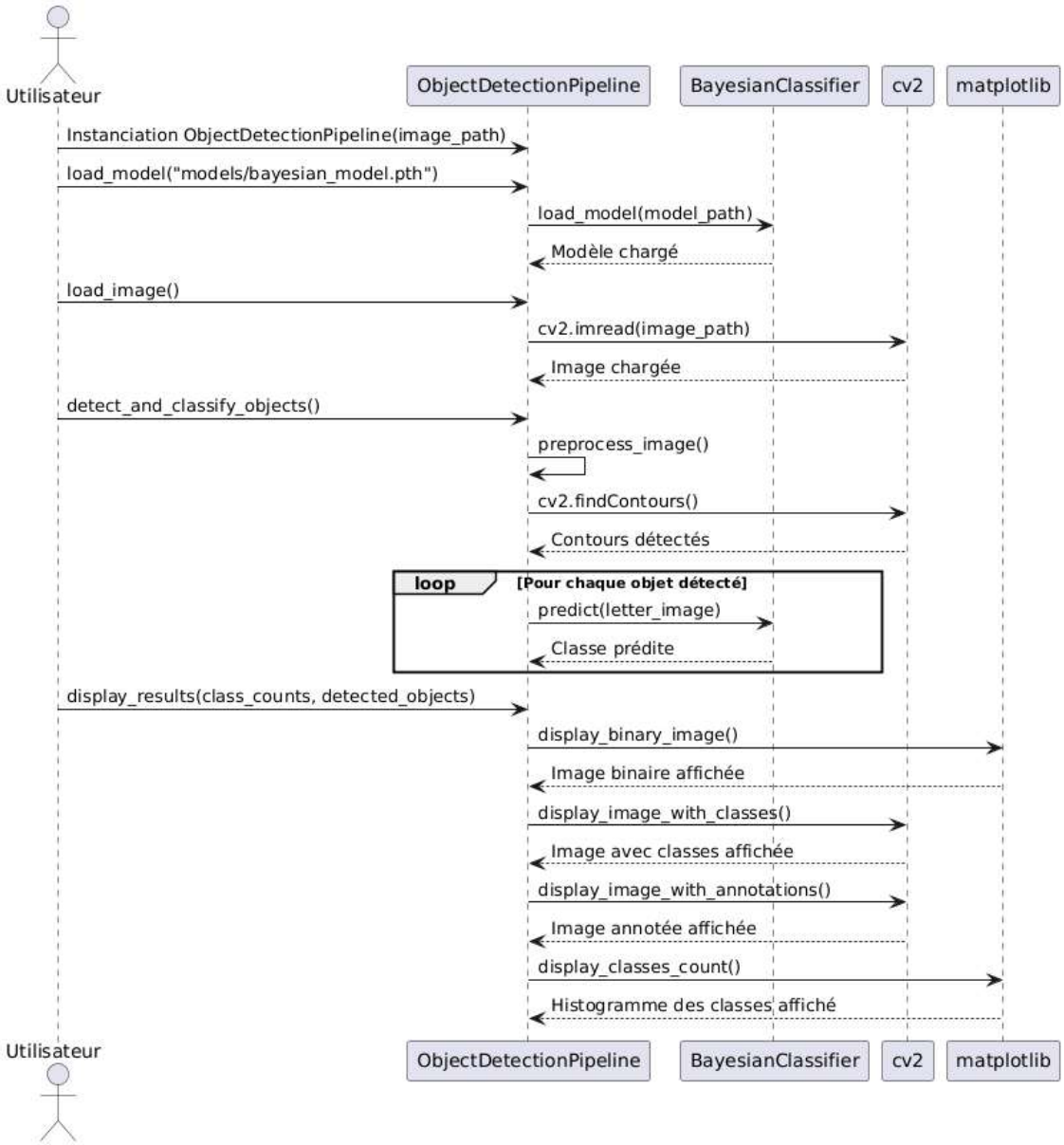
Chaque objet détecté est extrait et classifié individuellement par le modèle, qui prédit une classe pour chaque région d'intérêt.

Les résultats sont affichés sous plusieurs formes :

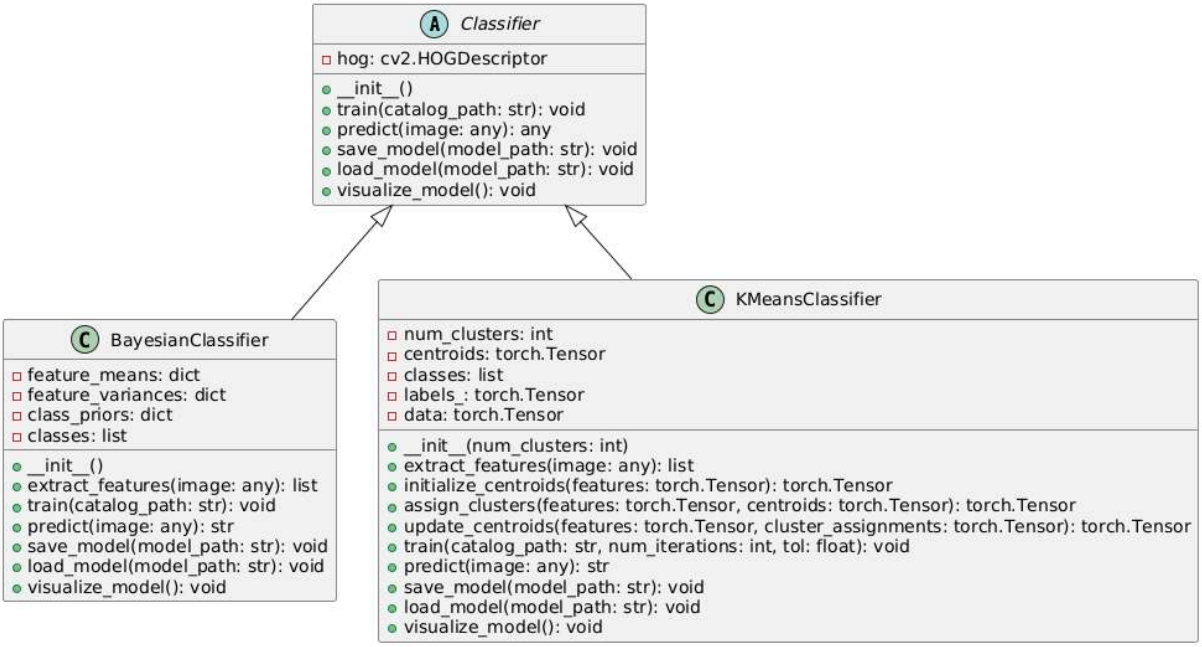
- **Image binaire**
- **Image avec classes prédites**
- **Image annotée avec les prédictions**
- **Histogramme** du nombre d'occurrences des classes détectées

Des informations supplémentaires sont également affichées dans la console.

Ce processus permet une évaluation visuelle et statistique des résultats de la classification et de l'identification des classes dans l'image choisie


Utilisateur
ObjectDetectionPipeline
BayesianClassifier
cv2
matplotlib
Instanciation ObjectDetectionPipeline(image_path)
load_model("models/bayesian_model.pth")
load_model(model_path)
Modèle chargé
load_image()
cv2.imread(image_path)
Image chargée
detect_and_classify_objects()
preprocess_image()
cv2.findContours()
Contours détectés
loop
[Pour chaque objet détecté]
predict(letter_image)
Classe prédite
display_results(class_counts, detected_objects)
display_binary_image()
Image binaire affichée
display_image_with_classes()
Image avec classes affichée
display_image_with_annotations()
Image annotée affichée
display_classes_count()
Histogramme des classes affiché
Utilisateur
ObjectDetectionPipeline
BayesianClassifier
cv2
matplotlib

6) Diagramme de classe des classificateurs



Cet héritage permet à la pipeline d'être utilisable par les deux classificateurs, que ce soit le Bayésien ou bien même le Kmeans.

Cette approche permet à la pipeline de gérer facilement différents types de classificateurs. Par exemple, un utilisateur peut choisir d'utiliser un modèle Bayésien pour certaines images et un modèle K-means pour d'autres, en fonction des besoins du projet, des performances attendues ou des contraintes techniques. L'utilisation de l'héritage facilite aussi l'ajout de nouvelles fonctionnalités de manière simple et sans perturber le reste du système, ce qui permet à chaque classificateur d'évoluer de façon indépendante sans impacter les autres parties de la pipeline.

Résultat des models

Model

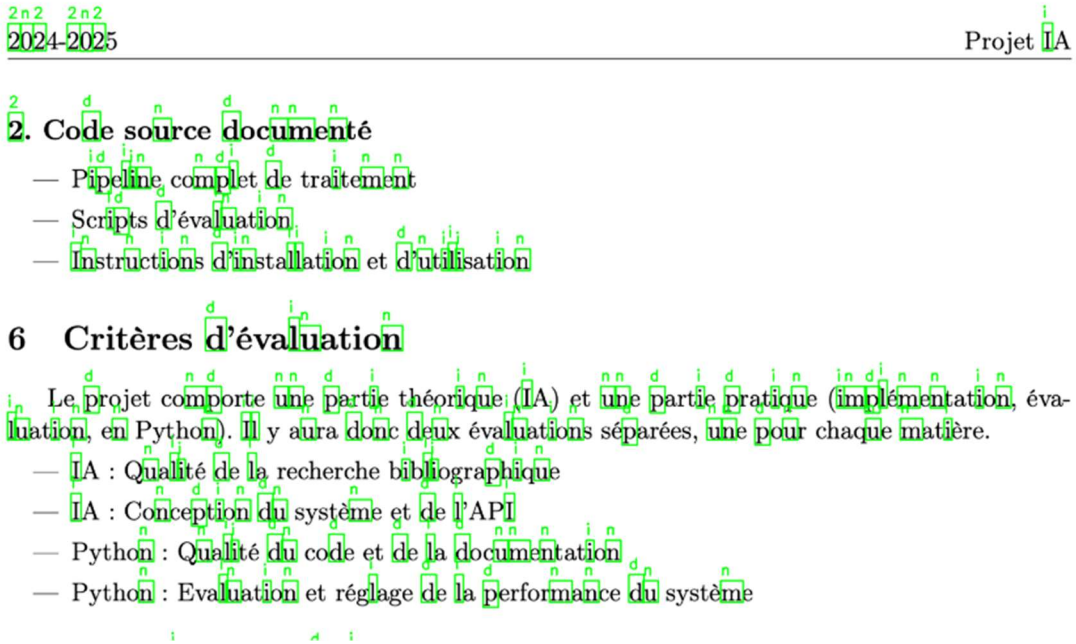
Bayésien : « Fonctionnel »

Test -100 000 (Plus le terme est proche de 0 plus il exige un taux de confiance élevé au contraire plus il est petit moins les classe affiché seront fiable):

```
# Retourner la classe ayant la probabilité
m = max(posteriors, key=posteriors.get)
if posteriors[m] < -100000:
    return None
return m
```

(Fichier produit en entrainement)

(Résultat du plan et du texte passé en paramètre) catalogue « réduit »: d i n 2


2024-2025 Projet IA

2. Code source documenté

— Pipeline complet de traitement
— Scripts d'évaluation
— Instructions d'installation et d'utilisation

6 Critères d'évaluation

Le projet comporte une partie théorique (IA) et une partie pratique (implémentation, évaluation, en Python). Il y aura donc deux évaluations séparées, une pour chaque matière.

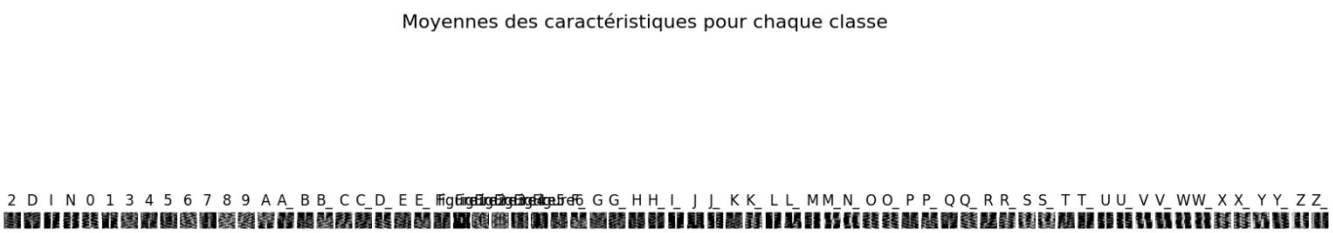
— IA : Qualité de la recherche bibliographique
— IA : Conception du système et de l'API
— Python : Qualité du code et de la documentation
— Python : Evaluation et réglage de la performance du système

Ici le catalogue a été limité à 4 caractères est dû à la rotation effectuer sur chaque caractère extrait du document, on peut constater que la classe « n » identifie les « u » dans sa classe, de même pour « d » qui assimile les « p » dans sa classe, cela est du au faite que c'est caractère ont une ressemblance élevée suite à une rotation à 180°.

Model Fini :

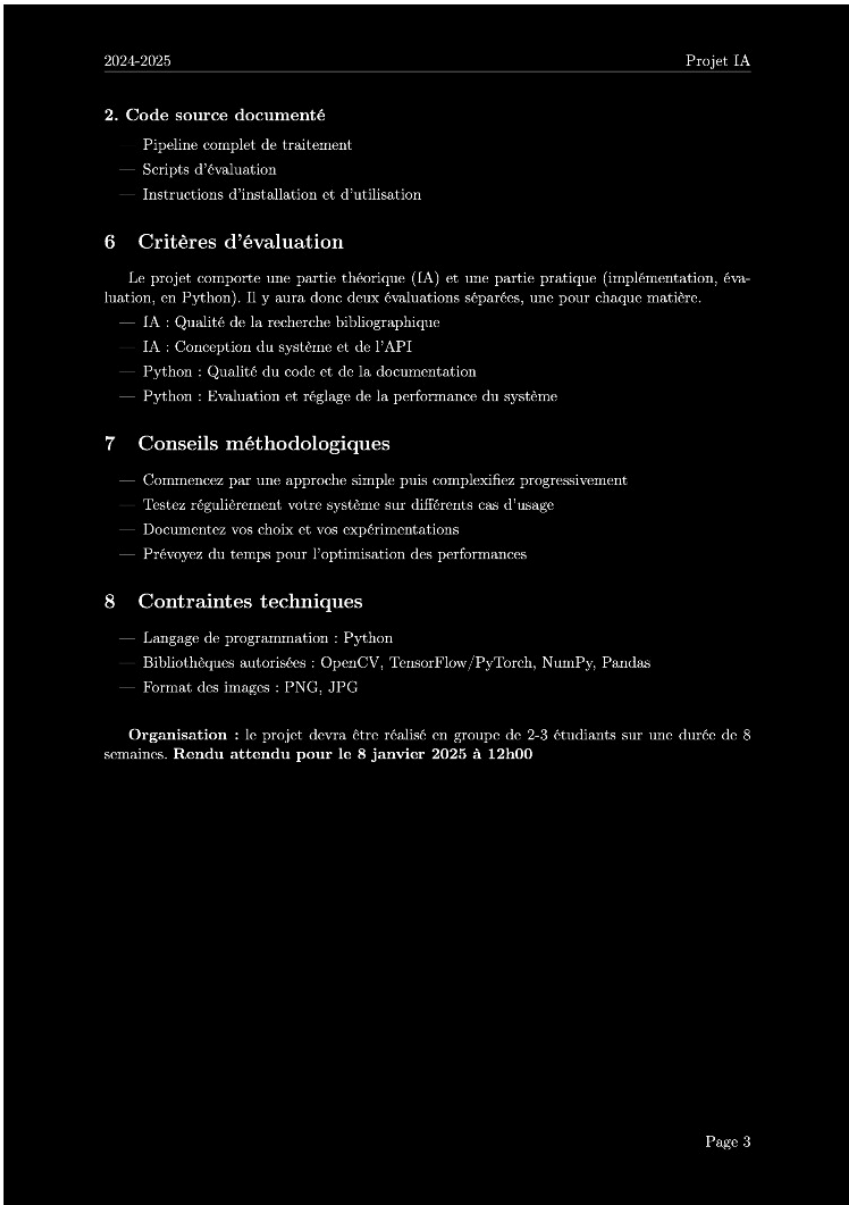
Parti entrainement :

Moyenne caractéristique de chaque classe



Parti utilisation du modèle :

Texte :



2024-2025 Projet IA

2. Code source documenté

- Pipeline complet de traitement
- Scripts d'évaluation
- Instructions d'installation et d'utilisation

6 Critères d'évaluation

Le projet comporte une partie théorique (IA) et une partie pratique (implémentation, évaluation, en Python). Il y aura donc deux évaluations séparées, une pour chaque matière.

- IA : Qualité de la recherche bibliographique
- IA : Conception du système et de l'API
- Python : Qualité du code et de la documentation
- Python : Evaluation et réglage de la performance du système

7 Conseils méthodologiques

- Commencez par une approche simple puis complexifiez progressivement
- Testez régulièrement votre système sur différents cas d'usage
- Documentez vos choix et vos expérimentations
- Prévoyez du temps pour l'optimisation des performances

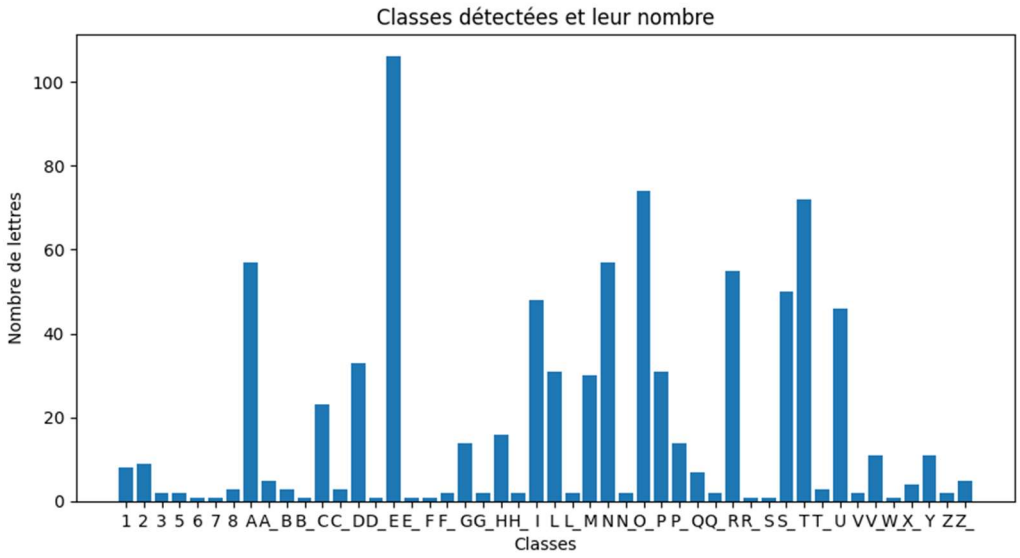
8 Contraintes techniques

- Langage de programmation : Python
- Bibliothèques autorisées : OpenCV, TensorFlow/PyTorch, NumPy, Pandas
- Format des images : PNG, JPG

Organisation : le projet devra être réalisé en groupe de 2-3 étudiants sur une durée de 8 semaines. **Rendu attendu pour le 8 janvier 2025 à 12h00**

Page 3

Page transformé fourni au modèle pour faciliter la détection des formes.



Occurrence d'élément par classe

2. Code source documenté

- Pipeline complet de traitement
- Scripts d'évaluation
- Instructions d'installation et d'utilisation

6 Critères d'évaluation

Le projet comporte une partie théorique (IA), et une partie pratique (implémentation, évaluation, en Python). Il y aura donc deux évaluations séparées, une pour chaque matière.

- IA : Qualité de la recherche bibliographique
- IA : Conception du système et de l'API
- Python : Qualité du code et de la documentation
- Python : Evaluation et réglage de la performance du système

7 Conseils méthodologiques

- Commencez par une approche simple puis complexifiez progressivement
- Testez régulièrement votre système sur différents cas d'usage
- Documentez vos choix et vos expérimentations
- Prévoyez du temps pour l'optimisation des performances

8 Contraintes techniques

- Langage de programmation : Python
- Bibliothèques autorisées : OpenCV, TensorFlow/PyTorch, NumPy, Pandas
- Format des images : PNG, JPG

Organisation : le projet devra être réalisé en groupe de 2-3 étudiants sur une durée de 8 semaines. Rendu attendu pour le 8 janvier 2025 à 12h00

Résultat : Chaque caractère identifié est annoté avec la classe auquel il a été rattaché

Statistique du modèle bayésien sur les classes identifiées :

```
Précision et Rappel :
1: Précision=1.0, Rappel=8.0, Find=8, Vrai=1
2: Précision=1.0, Rappel=1.0, Find=9, Vrai=9
3: Précision=1.0, Rappel=1.0, Find=2, Vrai=2
5: Précision=0.5, Rappel=0.5, Find=1, Vrai=2
6: Précision=1.0, Rappel=1.0, Find=1, Vrai=1
8: Précision=1.0, Rappel=1.0, Find=3, Vrai=3
A: Précision=0.96875, Rappel=0.96875, Find=62, Vrai=64
B: Précision=1.0, Rappel=1.0, Find=4, Vrai=4
C: Précision=0.7666666666666667, Rappel=0.7666666666666667, Find=23, Vrai=30
D: Précision=1.0, Rappel=1.0, Find=34, Vrai=34
E: Précision=0.781021897810219, Rappel=0.781021897810219, Find=107, Vrai=137
F: Précision=0.42857142857142855, Rappel=0.42857142857142855, Find=3, Vrai=7
G: Précision=1.0, Rappel=1.0, Find=16, Vrai=16
H: Précision=1.0, Rappel=1.125, Find=18, Vrai=16
I: Précision=0.7384615384615385, Rappel=0.7384615384615385, Find=48, Vrai=65
L: Précision=1.0, Rappel=1.0, Find=33, Vrai=33
M: Précision=1.0, Rappel=1.0, Find=30, Vrai=30
N: Précision=1.0, Rappel=1.0, Find=59, Vrai=59
P: Précision=1.0, Rappel=1.0, Find=45, Vrai=45
Q: Précision=1.0, Rappel=1.0, Find=9, Vrai=9
R: Précision=1.0, Rappel=1.0, Find=55, Vrai=55
T: Précision=1.0, Rappel=1.0, Find=75, Vrai=75
U: Précision=1.0, Rappel=1.0, Find=46, Vrai=46
V: Précision=1.0, Rappel=1.0, Find=13, Vrai=13
Y: Précision=1.0, Rappel=1.0, Find=11, Vrai=11
O: Précision=1.0, Rappel=1.0769230769230769, Find=70, Vrai=65
S: Précision=0.9803921568627451, Rappel=0.9803921568627451, Find=50, Vrai=51
W: Précision=1.0, Rappel=1.0, Find=1, Vrai=1
X: Précision=1.0, Rappel=1.0, Find=4, Vrai=4
Z: Précision=1.0, Rappel=1.0, Find=5, Vrai=5
Précision moyenne : 0.9387954562790866, Rappel moyen : 1.1788595588431892
```

On peut constater qu'à part le 1 qui est surreprésenté (rappel de 8) et le 5 et i qui sont sous-représenter (rappel de 0.5 et 0.73 respectivement) le reste des élément trouver possède un rappel de 1 ou en est très proche nous pouvons donc conclure que la confiance que l'ont peut accorder à cette IA pour reconnaitre les lettres (de cette police) est élevé au vu de ces performance.

Voici la réécriture du texte par les classes identifié (chaque caractère reconnu est effacé puit remplacer par le nom de sa classe) :

2. Code source documenté

- Pipeline complet de traitement
- Scripts d'évaluation
- Instructions d'installation et d'utilisation

6 Critères d'évaluation

Le projet comporte une partie théorique (IA) et une partie pratique (implémentation, évaluation, en Python). Il y aura donc deux évaluations séparées, une pour chaque matière.

- IA : Qualité de la recherche bibliographique
- IA : Conception du Système et de l'API
- Python : Qualité du code et de la documentation
- Python : Evaluation et réglage de la performance du Système

7 Conseils méthodologiques

- Commencez par une approche simple puis complexifiez progressivement
- Testez régulièrement votre Système sur différents cas d'usage
- Documentez vos choix et vos expérimentations
- Prévoyez du temps pour l'Optimisation des performances

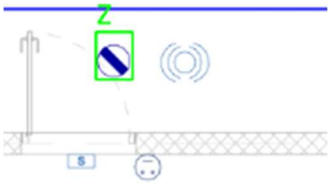
8 Contraintes techniques

- Langage de programmation : Python
- Bibliothèques autorisées : OpenCV, TensorFlow/PyTorch, NumPy, Pandas
- Format des images : PNG, rPG

Organisation : le projet devra être réalisé en groupe de 2-3 étudiants Sur une durée de 8 Semaines. Rendu attendu pour le 8 janvier 2025 à 12h00

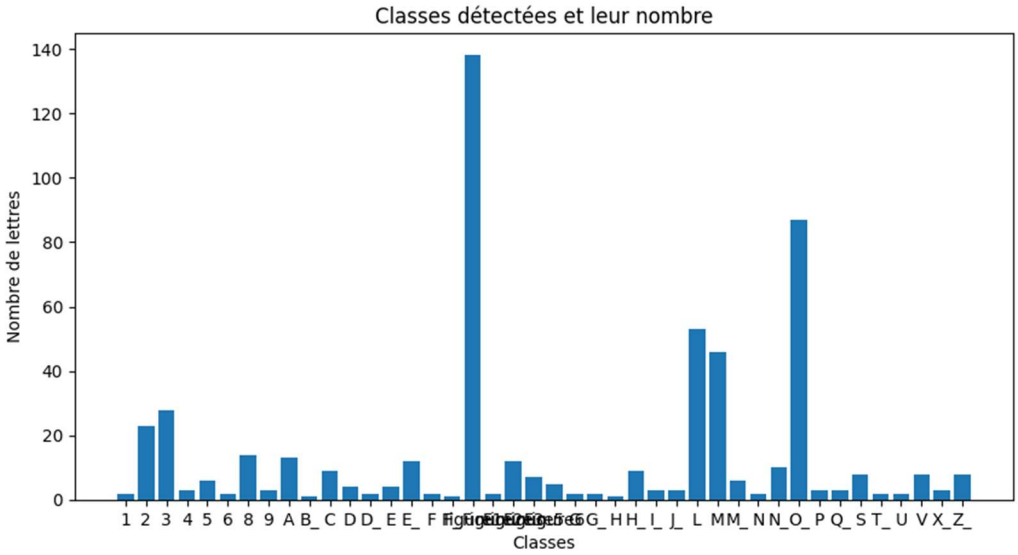
Plan :

Reconnais certaine figure et ce trompe de classe pour d'autre :



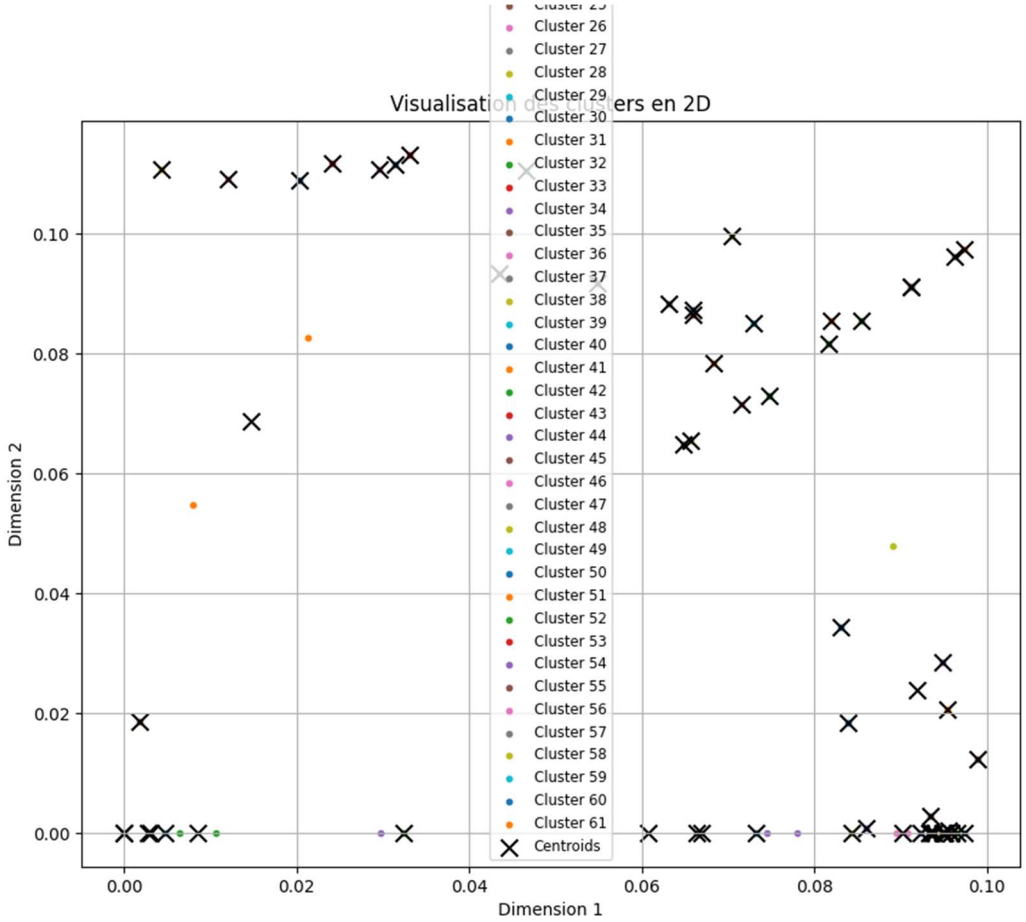
Le texte empêche la reconnaissance de figure par endroit :

Avec l'occurrence des figures et texte sur le plan :



Kmean : « Insatisfaisant »

Identification des centroide des 62 clusters (26 minuscule + 26 majuscule + 10 chiffre + 6 figure) :



On peut constater que plusieurs centroïde se superpose au lieu de s’espacer dans le graphe de la nous pouvons naturellement douter de la fiabilité de ce modèle.

2. Code source documenté

- Pipeline complet de traitement
- Scripts d'évaluation
- Instructions d'installation et d'utilisation

6 Critères d'évaluation

Le projet comporte une partie théorique (IA) et une partie pratique (implémentation, évaluation, en Python). Il y aura donc deux évaluations séparées, une pour chaque matière.

- IA : Qualité de la recherche bibliographique
- IA : Conception du système et de l'API
- Python : Qualité du code et de la documentation
- Python : Evaluation et réglage de la performance du système

7 Conseils méthodologiques

- Commencez par une approche simple puis complexifiez progressivement
- Testez régulièrement votre système sur différents cas d'usage
- Documentez vos choix et vos expérimentations
- Prévoyez du temps pour l'optimisation des performances

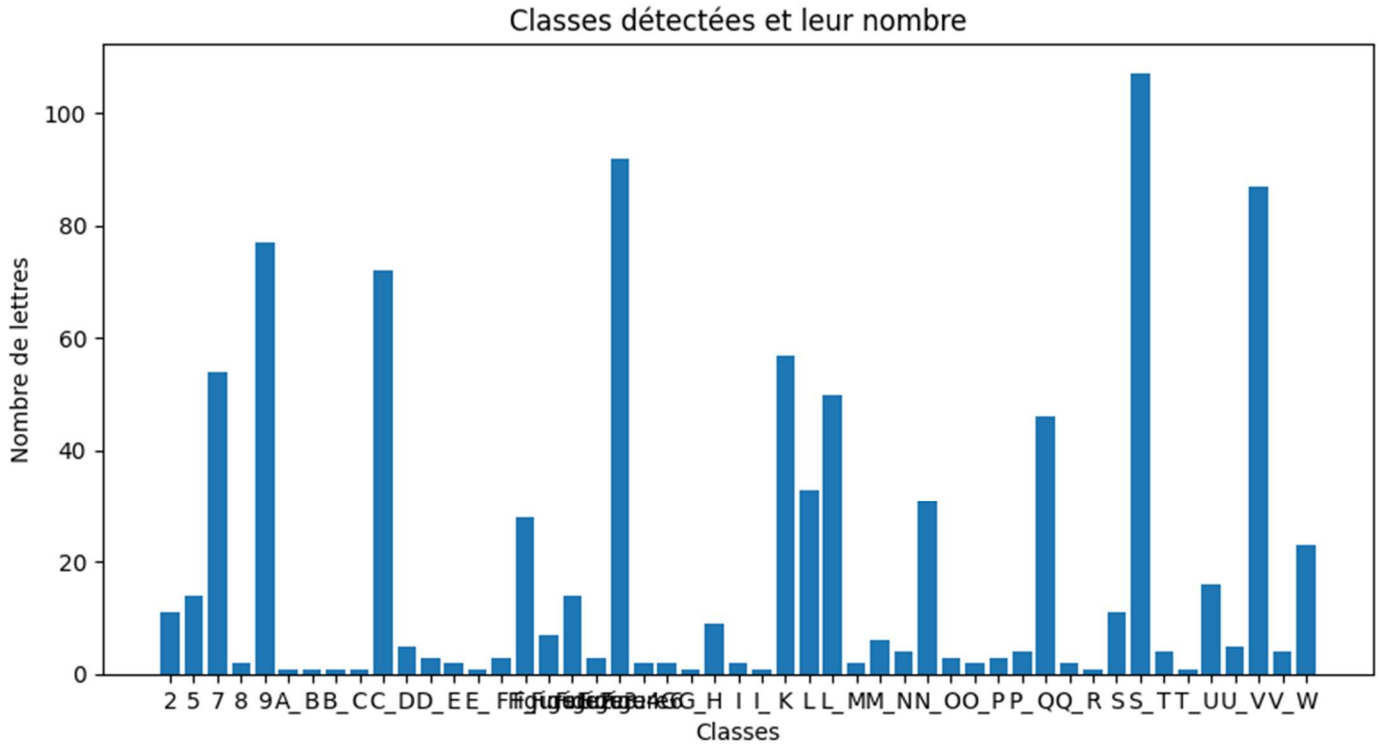
8 Contraintes techniques

- Langage de programmation : Python
- Bibliothèques autorisées : OpenCV, TensorFlow/PyTorch, NumPy, Pandas
- Format des images : PNG, JPG

Organisation : le projet devra être réalisé en groupe de 2-3 étudiants sur une durée de 8 semaines. **Rendu attendu pour le 8 janvier 2025 à 12h00**

Résultat : Chaque caractère identifié est annoté avec la classe auquel il a été rattaché

Occurrence d'élément par classe



```
Précision et Rappel :
2: Précision=1.0, Rappel=1.2222222222222223, Find=11, Vrai=9
5: Précision=1.0, Rappel=7.0, Find=14, Vrai=2
7: Précision=1.0, Rappel=54.0, Find=54, Vrai=1
8: Précision=0.6666666666666666, Rappel=0.6666666666666666, Find=2, Vrai=3
9: Précision=1.0, Rappel=0, Find=77, Vrai=0
B: Précision=0.5, Rappel=0.5, Find=2, Vrai=4
C: Précision=1.0, Rappel=2.433333333333333, Find=73, Vrai=30
D: Précision=0.23529411764705882, Rappel=0.23529411764705882, Find=8, Vrai=34
E: Précision=0.021897810218978103, Rappel=0.021897810218978103, Find=3, Vrai=137
F: Précision=1.0, Rappel=4.428571428571429, Find=31, Vrai=7
Figure1: Précision=1.0, Rappel=0, Find=7, Vrai=0
Figure2: Précision=1.0, Rappel=0, Find=14, Vrai=0
Figure3: Précision=1.0, Rappel=0, Find=3, Vrai=0
Figure4: Précision=1.0, Rappel=0, Find=92, Vrai=0
Figure6: Précision=1.0, Rappel=0, Find=2, Vrai=0
G: Précision=0.1875, Rappel=0.1875, Find=3, Vrai=16
H: Précision=0.5625, Rappel=0.5625, Find=9, Vrai=16
I: Précision=0.046153846153846156, Rappel=0.046153846153846156, Find=3, Vrai=65
K: Précision=1.0, Rappel=0, Find=57, Vrai=0
L: Précision=1.0, Rappel=2.515151515151515, Find=83, Vrai=33
M: Précision=0.26666666666666666, Rappel=0.26666666666666666, Find=8, Vrai=30
N: Précision=0.5932203389830508, Rappel=0.5932203389830508, Find=35, Vrai=59
O: Précision=0.07692307692307693, Rappel=0.07692307692307693, Find=5, Vrai=65
P: Précision=0.15555555555555556, Rappel=0.15555555555555556, Find=7, Vrai=45
Q: Précision=1.0, Rappel=5.333333333333333, Find=48, Vrai=9
R: Précision=0.01818181818181818, Rappel=0.01818181818181818, Find=1, Vrai=55
S: Précision=1.0, Rappel=2.3137254901960786, Find=118, Vrai=51
T: Précision=0.06666666666666667, Rappel=0.06666666666666667, Find=5, Vrai=75
U: Précision=0.45652173913043476, Rappel=0.45652173913043476, Find=21, Vrai=46
V: Précision=1.0, Rappel=7.0, Find=91, Vrai=13
W: Précision=1.0, Rappel=23.0, Find=23, Vrai=1
A: Précision=0.015625, Rappel=0.015625, Find=1, Vrai=64
Précision moyenne : 0.6521679157123068, Rappel moyen : 3.5348659570500542
```

Nous pouvons voir que les performance ce modèle sont médiocre ayant un rappel et une précision moyenne éloigné de 1.

Bibliographie

Danyang. (2023, Mai 19). *Comparison of HOG (Histogram of Oriented Gradients) and SIFT (Scale Invariant Feature Transform)*. Récupéré sur Medium: https://medium.com/@danyang95luck/comparison-of-hog-histogram-of-oriented-gradients-and-sift-scale-invariant-feature-transform-e2b17f61c9a3

OpenCV. (s.d.). Récupéré sur https://github.com/opencv/opencv/wiki

PyTorch. (s.d.). Récupéré sur https://pytorch.org

TensorFlow. (s.d.). Récupéré sur https://www.tensorflow.org

Wei, L., Dragomir, A., Dumitru, E., Christian, S., Scott, R., Cheng-Yang, F., & Alexander C., B. (s.d.). *SSD: Single Shot MultiBox Detector*. Récupéré sur Arxiv: https://arxiv.org/pdf/1512.02325

Pytorch vs TensorFlow. (2024, Oct 23). Vihar Kurama .”*PyTorch vs. TensorFlow: Key Differences to Know for Deep Learning”: https://builtin.com/data-science/pytorch-vs-tensorflow#:~:text=PyTorch%20is%20ideal%20for%20research,high-performance%20and%20scalable%20models*.

Machine Learning vs Deep Learning, (2024, Nov 10). Richard Gastard, ” Machine Learning vs Deep Learning : quelles différences? ”: https://www.jedha.co/formation-ia/vraie-difference-machine-learning-deep-learning

HOG vs SIFT, (May 19, 2023),« Comparison of HOG (Histogram of Oriented Gradients) and SIFT (Scale Invariant Feature Transform) »,https://medium.com/@danyang95luck/comparison-of-hog-histogram-of-oriented-gradients-and-sift-scale-invariant-feature-transform-e2b17f61c9a3

Annexes

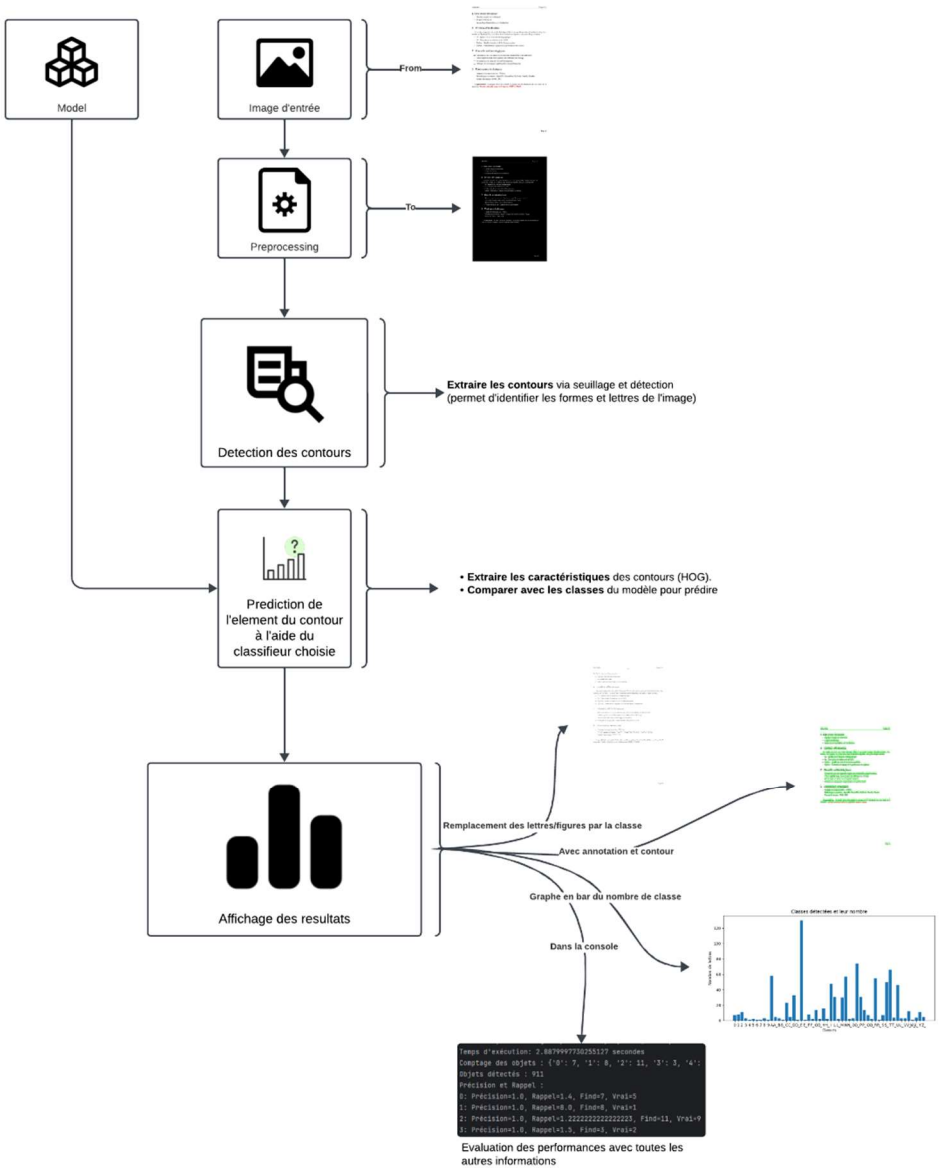


Figure 1: Utilisation d'un model sur une image

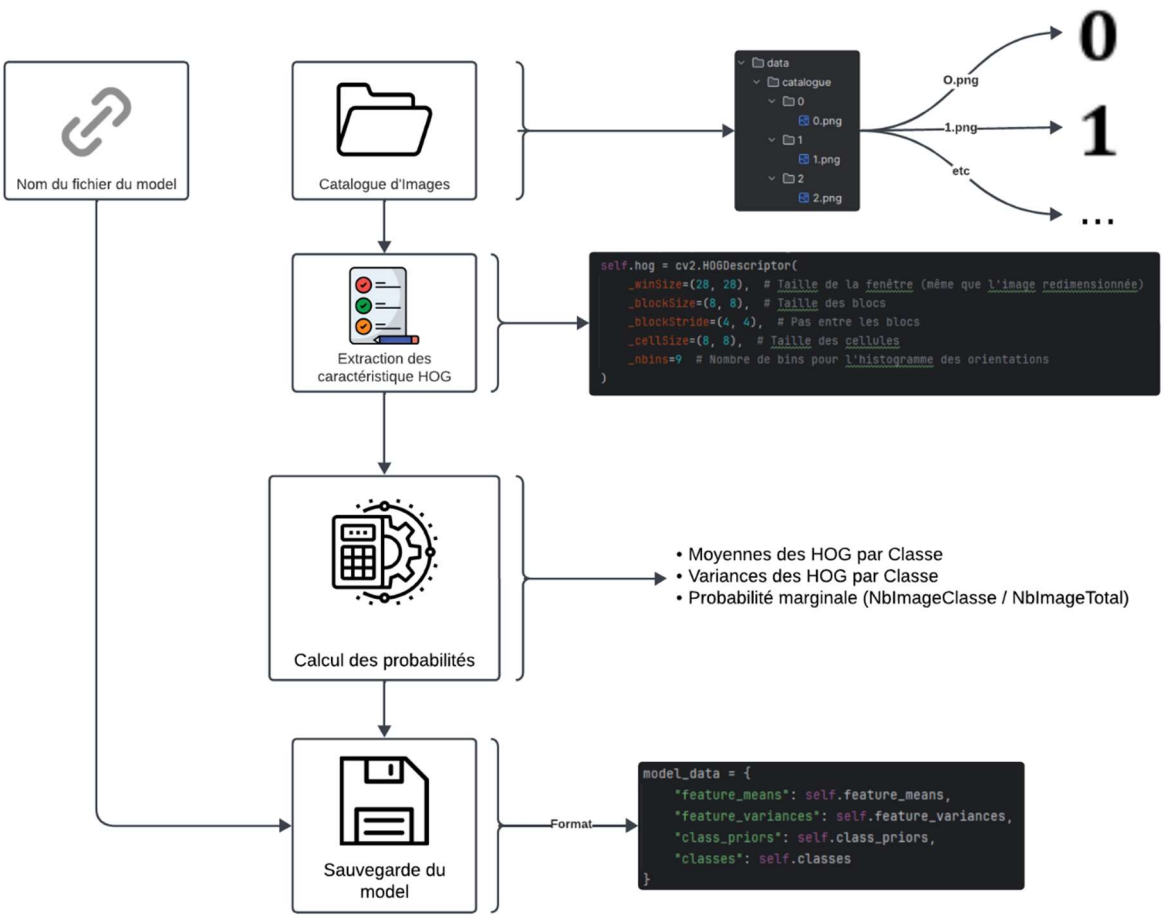


Figure 2: Entrainement du model bayésien

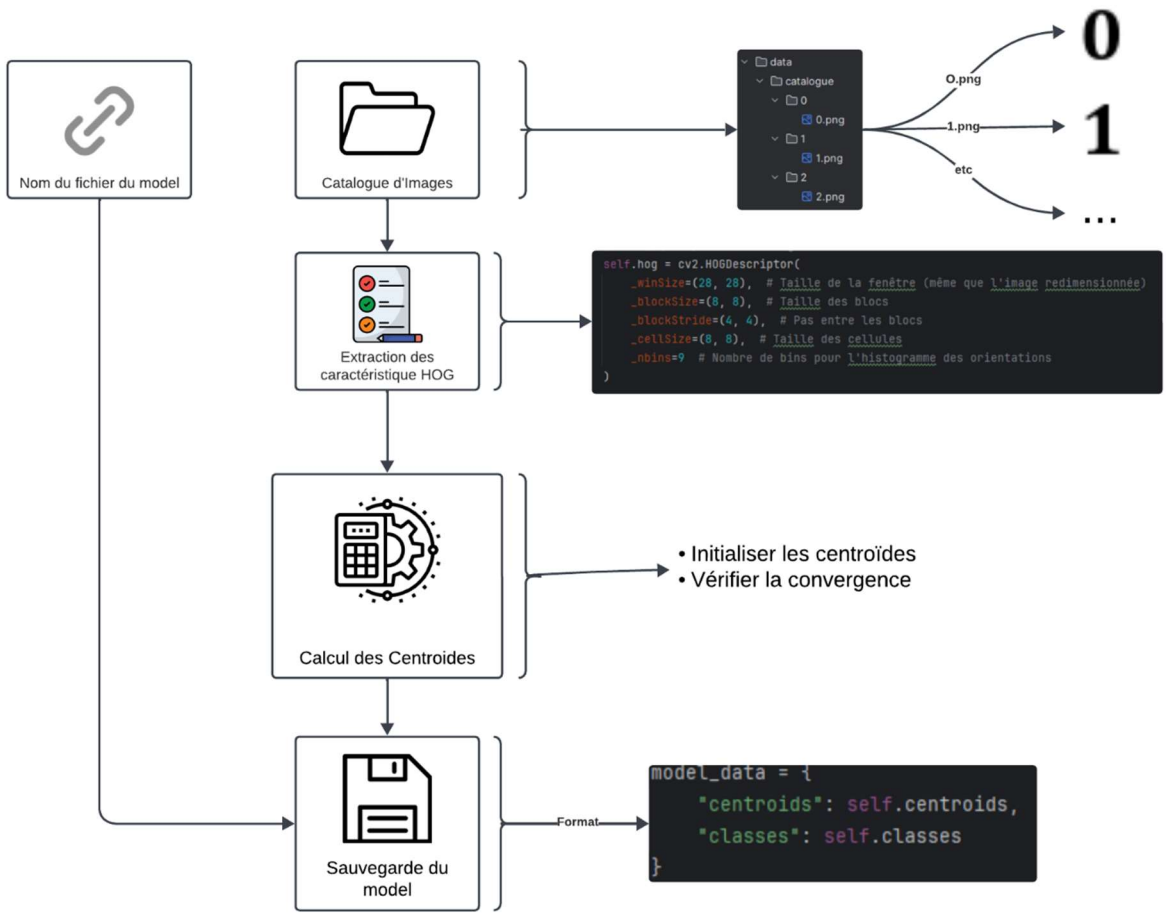


Figure 3:Entrainement du modèle Kmean