الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

---

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

# Université d'Algiers 1 - Benyoucef Benkhadda

**Faculté des Sciences**

**Département Informatique**

**1ère année Master ISII**

**Module :** Traitement d'Images Numérique

## Projet :

# Génération de descriptions d'images avec des modèles d'apprentissage automatique


**Réaliser par :**

| | | | |
|---|---|---|---|
| Boukaiou Sabrine | grp 2 | Barkat Youcef | grp 2 |
| Kessar Meriem | grp 2 | Boulaloua Hiba | grp 2 |
| Daoudi Mohamed Samy | grp 2 | Belahcene Samy | grp 1 |

# Table des matières

**Introduction Général** ..... **1**

1. Contexte et Problématique ..... 1
2. Objectif de l'étude ..... 1
3. Méthodologie Générale ..... 1
4. Outils et Bibliothèques Utilisés ..... 2

**Chapitre 1 : Prétraitement des données** ..... **2**

1. Introduction ..... 2
2. Configuration initiale et chargement de données ..... 2
3. Prétraitement ..... 3
   - 3.1. Prétraitement des images ..... 3
   - 3.2. Prétraitement des captions ..... 4
4. Extraction des caractéristiques SIFT ..... 6
5. Construction d'un dictionnaire visuel : BoVW ..... 7
6. Reduction de dimensions PCA ..... 7

**Chapitre 2 : Modèles de Classification** ..... **8**

1. Introduction ..... 8
2. Partie commune ..... 8
3. Naive Bayes ..... 9
4. Knn (K-NEAREST-NEIGHBORS) ..... 10
5. Gradient Boosting ..... 10
6. Random Forest ..... 12
7. MLPClassifier ..... 13
8. Decision Trees ..... 14

**Chapitre 3 : Segmentation et Généralisation d'une phrase Gabarit** ..... **15**

1. Génération de phrase Gabarit ..... 15

2. Approche de Segmentation ..... 15

2. Approche de Segmentation ..... 15

**Chapitre 4 : Comparaison et Analyse** ..... **17**

1. Métriques d'Évaluation ..... 17
2. Résultats de Tous les Modèles ..... 17
3. Classement ..... 18

4. Decision Tree vs Autres Modèles ..... 19
5. Conclusion ..... 19
**Conclusion Générale ..... 19**

# Introduction Général

## 1. Contexte et Problématique

L'annotation automatique d'images représente un défi majeur en vision par ordinateur, visant à combler le fossé sémantique entre la représentation numérique des images et leur interprétation humaine.

Dans un contexte d'explosion des données visuelles numériques, la génération automatique de descriptions pertinentes devient essentielle pour l'indexation, la recherche et l'accessibilité des contenus multimédia.

## 2. Objectif de l'étude

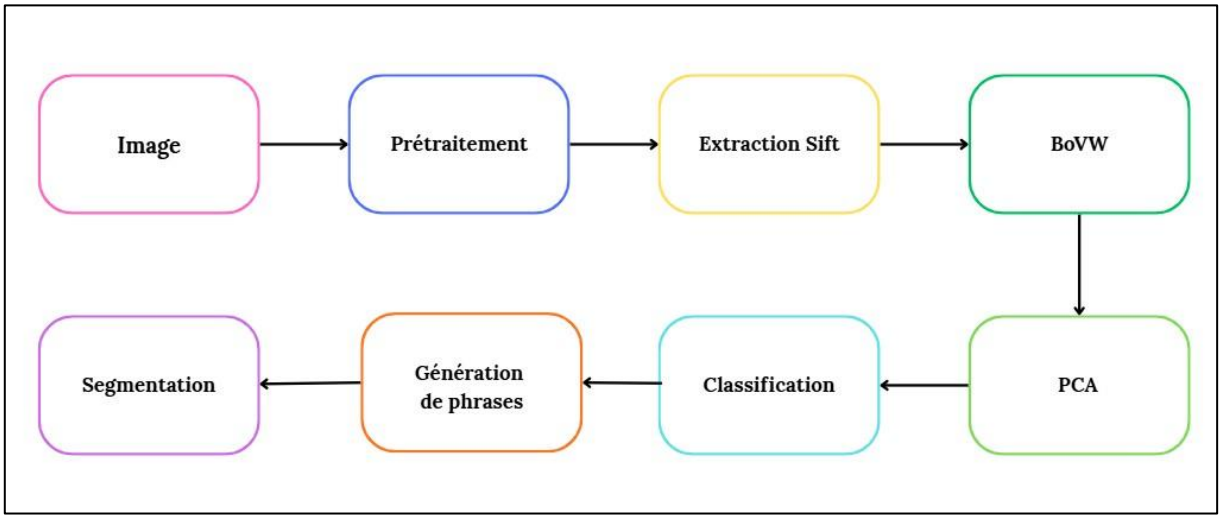
Ce projet vise à développer un système complet de génération de descriptions d'images en utilisant exclusivement des techniques classiques d'apprentissage automatique.

Les objectifs spécifiques sont :

1. Implémenter une chaine de traitement complète depuis l'image brute jusqu'à la description textuelle.
2. Comparer les performances de six algorithmes de classification différents.
3. Évaluer la qualité des descriptions générées.
4. Segmenter les objets détectés pour une analyse plus fine.

## 3. Méthodologie Générale

Notre approche suit un pipeline structuré en cinq étapes principales :



## 4. Outils et Bibliothèques Utilisés

✓ **Jeux de données :**

- **Flickr8k dataset :** https://www.kaggle.com/datasets/adityajn105/flickr8k

✓ **Langage :** Python

✓ **Bibliothèques :**

- **Gestion/calcul :** os, numpy, pandas, matplotlib.pyplot
- **Images/texte :** cv2, re, string, collections.Counter
- **Nettoyage texte** : sklearn.feature_extraction.text.ENGLISH_STOP_WORDS
- **Machine Learning** : StandardScaler, MiniBatchKMeans, PCA, MultiLabelBinarizer, sklearn.multiclass, sklearn.model_selection, sklearn.metrics, sklearn.neural_network, sklearn.ensemble, sklearn.naive_bayes, sklearn.neighbors, sklearn.tree
- **Sérialisation** : pickle

✓ **Github :**

- **Lien du projet** : https://github.com/sabrinebk/TIN_Boukaiou_Kessar_Boulaloua_Daoudi_Belahcene_Barkat

# Chapitre 1 : Prétraitement des données

## 1. Introduction

Ce chapitre décrit les étapes de prétraitement des données pour le projet de génération de descriptions d'images. Nous présentons le traitement des images, le nettoyage des captions, l'extraction de caractéristique SIFT, la construction du dictionnaire visuel BoVW et la réduction de dimension PCA

## 2. Configuration initiale et chargement de données

### Structure du dataset :

Le Dataset utilisé est **Flickrk8,** contenant :

- 8091 images
- Un fichier captions.txt avec 5 captions par image

**Exploration du dataset :**

```
os.listdir("/kaggle/input")
os.listdir("/kaggle/input/flickr8k")
```

```
['captions.txt', 'Images']
```

**Import**

```
import numpy as np
import pickle
import time
import warnings
import pandas as pd
warnings.filterwarnings("ignore")

from sklearn.preprocessing import StandardScaler, MultiLabelBinarizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import (
    accuracy_score, f1_score, precision_score, recall_score, hamming_loss,
    classification_report, multilabel_confusion_matrix
)
```

**Path**

```
# Dataset paths
DATASET_DIR = "/kaggle/input/flickr8k"
IMAGE_DIR = os.path.join(DATASET_DIR, "Images")
CAPTION_FILE = os.path.join(DATASET_DIR, "captions.txt")

# Output paths
OUTPUT_DIR = "/kaggle/working"

print("Images:", len(os.listdir(IMAGE_DIR)))
print("Caption file exists:", os.path.exists(CAPTION_FILE))
```

# 3. Prétraitement

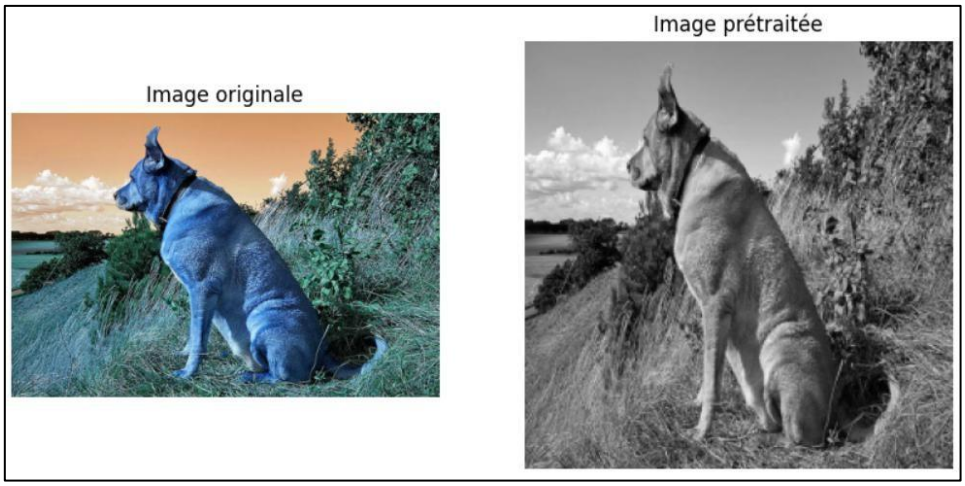
## 3.1. Prétraitement des images

Les images ont été chargées et uniformisées via un pipeline de traitement standardisées.

Chaque image a été convertie en niveaux de gris, redimensionnée à une dimension fixe de 256×256 pixels, puis normalisée dans l'intervalle [0-1] afin d'optimiser l'extraction ultérieure des descripteurs SIFT.

```python
def preprocess_image(image_path, target_size=TARGET_SIZE, update_stats=True):
    """Prétraite une image unique"""
    try:
        # 1. Chargement de l'image
        image = cv2.imread(image_path)
        if image is None:
            print(f"Erreur de chargement: {image_path}")
            if update_stats:
                image_stats['failed'] += 1
            return None

        # 2. Conversion en niveaux de gris (pour SIFT)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # 3. Redimensionnement
        resized = cv2.resize(gray, target_size, interpolation=cv2.INTER_AREA)

        # 4. Normalisation des intensités [0, 255] -> [0, 1]
        normalized = resized / 255.0

        if update_stats:
            image_stats['total_images'] += 1
            image_stats['successful'] += 1

        return normalized
```

**Test de Prétraitement sur une image**




## 3.2. Prétraitement des captions

Dans le processus de nettoyage des captions, plusieurs étapes ont été mise en œuvre. Afin d'éliminer les mots vides (stop words), nous avons importé et utilisé la liste prédéfinie de mots vides en anglais disponible dans la bibliothèque :

from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

```python
STOP_WORDS = set(ENGLISH_STOP_WORDS)

def clean_caption(caption):
    # 1. Conversion en minuscules
    caption = caption.lower()

    # 2. Suppression des caractères non alphabétiques
    caption = re.sub(r"[^a-z\s]", "", caption)

    # 3. Suppression des espaces multiples
    caption = re.sub(r'\s+', ' ', caption).strip()

    # 4. Tokenization simple avec split
    tokens = caption.split()

    # 5. Suppression des stopwords
    tokens = [w for w in tokens if w not in STOP_WORDS]

    # 6. Suppression single characters
    tokens = [w for w in tokens if len(w) > 1]
    tokens = ' '.join(tokens)

    return tokens
```

**Extraction des mots-clés**

```python
def extract_keywords(captions, min_keywords=3, max_keywords=5):
    """
    Extrait entre 3 et 5 mots-clés par image
    avec une pondération TF-IDF simple
    """

    cleaned_captions = [clean_caption(cap) for cap in captions]

    all_words = []
    for cleaned_cap in cleaned_captions:
        all_words.extend(cleaned_cap.split())

    # Calculer TF (Term Frequency)
    word_counts = Counter(all_words)
    total_words = len(all_words)

    # Score TF-IDF simple (avec IDF)
    doc_freq = {}
    for cleaned_cap in cleaned_captions:
        unique_words = set(cleaned_cap.split())
        for word in unique_words:
            doc_freq[word] = doc_freq.get(word, 0) + 1

    # Score TF × IDF
    keywords_with_scores = []
    for word, tf in word_counts.items():
        tf_score = tf / total_words
        idf_score = np.log((len(cleaned_captions) + 1) / (doc_freq[word] + 1) ) + 1
        final_score = tf_score * idf_score  # Poids IDF
        keywords_with_scores.append((word, final_score))

    # Trier par score
    keywords_with_scores.sort(key=lambda x: x[1], reverse=True)

    # Nombre final de mots (entre 3 et 5)
    n_words = min(max_keywords, max(min_keywords, len(keywords_with_scores)))

    return [word for word, score in keywords_with_scores[:n_words]]
```

Après nettoyage de toutes les captions, celles-ci sont groupées par image. Les mots-clés sont ensuite extraits (entre 3 et 5 par image) en utilisant une pondération TF-IDF simplifiée. Voici le résultat obtenu (5 premiers images) :

```
=== MOTS-CLÉS PAR IMAGE ===
image
1000268201_693b08cb0e.jpg       [girl, climbing, wooden, little, pink]
1001773457_577c3a7d70.jpg            [dog, black, road, dogs, spotted]
1002674143_1b742ab4b8.jpg    [girl, rainbow, little, painted, sitting]
1003163366_44323f5815.jpg              [man, bench, dog, lays, white]
1007129816_e794419615.jpg         [man, hat, glasses, orange, wearing]
Name: clean_caption, dtype: object
```

### Construction du vocabulaire

Le vocabulaire est construit à partir des mots-clés extraits, filtrés avec une fréquence minimale de 2 occurrences. Ce seuil permet d'éliminer les mots trop rares et réduire la dimensionnalité.

```
Taille du vocabulaire : 1557
Exemples de mots du vocabulaire : ['talking', 'door', 'jet', 'lies', 'broom', 'mask', 'pulling',
'guitar', 'poses', 'held', 'teeth', 'target', 'phone', 'shot', 'stick', 'letters', 'security',
'campsite', 'waterfall', 'professional']
```

## 4. Extraction des caractéristiques SIFT

L'algorithme SIFT extrait des caractéristiques visuelles robustes et discriminantes à partir des images prétraitées. Il détecte des points d'intérêts invariantes aux changements d'échelle et de motifs, puis génère pour chacun un descripteur de 128 dimensions. Ces descripteurs captent les motifs locaux de gradients et forment une signature numérique unique pour chaque région d'intérêt. Cette représentation vectorielle servira de base à la construction du dictionnaire visuel dans l'approche Bag of Visual Words.

```python
descriptors_per_image = []
valid_image_names = []

# Configuration SIFT
SIFT_N_FEATURES = 0  # 0 = illimité
SIFT_CONTRAST_THRESHOLD = 0.04
show_progress = True

# Créer le détecteur SIFT
sift_detector = cv2.SIFT_create(
    nfeatures=SIFT_N_FEATURES,
    contrastThreshold=SIFT_CONTRAST_THRESHOLD
)

# Mettre à jour le nombre total d'images
sift_stats['total_images'] = len(preprocessed_images)

# Boucle principale d'extraction
iterator = tqdm(zip(image_names, preprocessed_images), total=len(image_names)) if show_progress
else zip(image_names, preprocessed_images)

for img_name, image in iterator:
    try:
        # RECONVERSION OBLIGATOIRE uint8 pour SIFT
        image_uint8 = (image * 255).astype(np.uint8)

        # Détection des keypoints et calcul des descripteurs
        keypoints, descriptors = sift_detector.detectAndCompute(image_uint8, None)

    # Traitement des résultats
    if descriptors is not None :
        descriptors_per_image.append(descriptors)
        valid_image_names.append(img_name)

        sift_stats['successful'] += 1
        sift_stats['total_descriptors'] += len(descriptors)

        if sift_stats['descriptor_shape'] is None:
            sift_stats['descriptor_shape'] = descriptors.shape[1]
    else:
        print(f"Aucun descripteur trouvé pour: {img_name}")
        sift_stats['failed'] += 1

except Exception as e:
    print(f"Erreur pour {img_name}: {e}")
    sift_stats['failed'] += 1
```

- Résultat d'extraction réussie avec SIFT :

```
100%| | 8091/8091 [02:14<00:00, 60.24it/s]

=== STATISTIQUES SIFT ===
Total d'images: 8091
Extraction réussie: 8091
Extraction échouée: 0
Taux de succès: 100.00%
Descripteurs totaux: 3997834
Moyenne de descripteurs par image: 494.1
Dimension des descripteurs: 128

Extraction terminée. 8091 images ont des descripteurs SIFT.
```

## 5. Construction d'un dictionnaire visuel : BoVW

Dans cette étape, on crée un dictionnaire visuel de 300 mots (clusters K-means) à partir des descripteurs d'images, puis transforme chaque image en un histogramme de ces mots visuels.

```
Descriptors used for K-Mean :  (200000, 128)
▾ MiniBatchKMeans
MiniBatchKMeans(batch_size=10000, n_clusters=300, n_init=10, random_state=42)

BoVW shape: (8091, 300)
```

## 6. Reduction de dimensions PCA

La réduction de dimension PCA permet de compresser l'information contenue dans les descripteurs SIFT de 128 dimensions vers un espace de dimension réduite, tout en préservant la majeure partie de la variance.

```python
PCA_COMPONENTS = 100

pca = PCA(
    n_components=PCA_COMPONENTS,
    random_state=RANDOM_STATE
)

X_pca = pca.fit_transform(X_bovw)

print("PCA shape:", X_pca.shape)
```

Le modèle PCA conserve **77.18% de la variance totale** des données originales après réduction de dimensionnalité.

```python
explained_variance = np.sum(pca.explained_variance_ratio_)
print("Explained variance:", explained_variance)
```

```
Explained variance: 0.7717797495579473
```

# Chapitre 2 : Modèles de Classification

## 1. Introduction

Ce chapitre présente l'évaluation comparative de plusieurs modèles de classification pour la prédiction de mots-clés d'images. Chaque modèle sera optimisé par validation croisée et Grid Search, puis appliqué selon une approche One-vs-Rest, traitant chaque mot-clé comme une classe à prédire.

## 2. Partie commune

- Avant de passer à chaque modèle de classification, l'importation des bibliothèques nécessaires dans le processus et l'importation des données est effectuée. Les fichiers prétraités contenant les caractéristiques (X), les étiquettes (Y) et les noms des classes sont chargé en mémoire, fournissant les données d'entrée pour l'étape de modélisation.

```
# ------------------------------
# 1 Chargement des données
# ------------------------------
X = np.load("/kaggle/input/dataset-pre-traitement-sift-bovw-pca/X_pca.npy")
y = np.load("/kaggle/input/dataset-pre-traitement-sift-bovw-pca/y.npy")

with open("/kaggle/input/dataset-pre-traitement-sift-bovw-pca/label_names.pkl", "rb") as f:
    label_names = pickle.load(f)
```

- Avant l'entraînement des modèles, les données sont standardisées pour assurer une échelle commune entre les caractéristiques, puis divisées en ensemble d'entraînement (80%) et de test (20%) avec une graine fixe pour la reproductibilité.

```
# ------------------------------
# 2 Standardisation
# ------------------------------
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ------------------------------
# 3 Division Train / Test
# ------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

print(f" X_train : {X_train.shape} | X_test : {X_test.shape}")
print(f" y_train : {y_train.shape} | y_test : {y_test.shape}")
```

- L'évaluation de chaque modèle de classification est réalisée en calculant plusieurs métriques de performance (Accuracy, Precision_Micro, Recall_micro, F1_micro / macro et Hamming_Loss) :

```python
# ------------------------------
# 6 Évaluation
# ------------------------------
y_pred = best_nb.predict(X_test)

metrics = {
    "Accuracy": accuracy_score(y_test, y_pred),
    "Precision_micro": precision_score(y_test, y_pred, average="micro", zero_division=0),
    "Recall_micro": recall_score(y_test, y_pred, average="micro", zero_division=0),
    "F1_macro": f1_score(y_test, y_pred, average="macro", zero_division=0),
    "F1_micro": f1_score(y_test, y_pred, average="micro", zero_division=0),
    "Hamming_Loss": hamming_loss(y_test, y_pred)
}

print("\n MÉTRIQUES DU MODÈLE Naive Bayes - MULTINOMIAL NB :")
for k, v in metrics.items():
    print(f"{k:20s} : {v:.4f}")
```

## 3. Naive Bayes

Le modèle Naive Bayes existe sous plusieurs formes, notamment : GaussianNB, MultinominalNB, BernoulliNB. Nous avons sélectionné le MultinominalNB car il convient bien aux caractéristiques discrètes ou de comptage issu de méthodes comme Bag of Words, et est efficace pour les problèmes de classification multi-classe avec des données creuses.

- Le modèle est le seul parmi tous les modèles utilisés qu'il a une standardisation spéciale utilisant :

```python
X = np.maximum(X, 0)
```

- Il est configuré pour la classification multi-classe via One-vs-Rest. Une recherche par grille sur le paramètre de lissage alpha est exécutée avec une validation croisée à 8 plis, optimisant le score F1-macro.

```python
# ------------------------------
# 4 Configuration MultinomialNB
# ------------------------------
nb_model = OneVsRestClassifier(
    MultinomialNB()
)

param_grid = {
    "estimator__alpha": [
        1e-4, 1e-3, 1e-2, 0.1, 0.5, 1.0
    ]
}

grid_search = GridSearchCV(
    estimator=nb_model,
    param_grid=param_grid,
    scoring="f1_macro",
    cv=8,
    verbose=2,
    n_jobs=1,
    refit=True
)
```

- L'évaluation du modèle :

```
 MÉTRIQUES DU MODÈLE Naive Bayes - MULTINOMIAL NB :
Accuracy             : 0.0000
Precision_micro      : 0.0429
Recall_micro         : 0.2064
F1_macro             : 0.0273
F1_micro             : 0.0710
Hamming_Loss         : 0.0475
```

**Observation :** Les performances très faibles suggèrent que la distribution du modèle Naive Bayes multinomiale n'est pas adaptée aux caractéristiques visuelles extraites (SIFT + PCA). La méthode One-vs-Rest appliquée sur un problème multi-classe potentiellement déséquilibré accentue ses limites. Une transformation des features ou un modèle plus flexible serait nécessaire.

## 4. Knn (K-NEAREST-NEIGHBORS)

Le K-Nearest Neighbors (KNN) est un algorithme de classification basé sur la proximité spatiale entre les points. Il fonctionne sur le principe qu'un point est classé selon la classe majoritaire de ses k voisins les plus proches dans l'espace des caractéristiques. Pour gérer la classification multi_label nous utilisons One-Vs-Rest.

Configuration du Modèle KNN :

```
# ------------------------------
# 4 Configuration Knn
# ------------------------------
# Modèle KNN de base
base_knn = KNeighborsClassifier(n_jobs=1)
# OneVsRest pour gérer la classification multi-label
knn_model = OneVsRestClassifier(base_knn)

param_grid = [{
    "estimator__n_neighbors": [5, 7],
    "estimator__weights": ["distance"],
    "estimator__metric": ["euclidean"],
    "estimator__p": [1, 2],
    "estimator__algorithm": ["brute"]
}]

grid_search = GridSearchCV(
    estimator=knn_model,
    param_grid=param_grid,
    cv=8,
    scoring="f1_macro",
    verbose=2,
    n_jobs=1,
    refit=True
)
```

L'évaluation du modèle :

```
 MÉTRIQUES DU MODÈLE KNN :
Accuracy            : 0.0006
Precision_micro     : 0.2235
Recall_micro        : 0.0162
F1_macro            : 0.0015
F1_micro            : 0.0302
Hamming_Loss        : 0.0091
```

## 5. Gradient Boosting

Le Gradient Boosting est une méthode d'apprentissage d'ensemble qui construit plusieurs apprenants faibles (généralement des arbres de décision) de manière séquentielle. Chaque nouvel apprenant est entraîné pour corriger les erreurs commises par les précédents, ce qui donne un modèle prédictif puissant. Cette approche itérative minimise une fonction de perte par optimisation du gradient.

**Application à la classification multi-étiquette :**

En classification multi-étiquette, chaque échantillon peut avoir plusieurs étiquettes cibles simultanément. Par exemple, une image peut être étiquetée avec à la fois « chien » et « extérieur ».

Configuration du Modèle Gradient Boosting :

```python
gb_classifier = GradientBoostingClassifier(
    random_state=42,
    verbose=0
)

ovr_model = OneVsRestClassifier(gb_classifier, n_jobs=-1)

param_grid = {
    'estimator__n_estimators': [200],
    'estimator__max_depth': [6],
    'estimator__learning_rate': [0.08]
}

f1_scorer = make_scorer(f1_score, average='samples', zero_division=0)

grid_search = GridSearchCV(
    estimator=ovr_model,
    param_grid=param_grid,
    scoring=f1_scorer,
    cv=5,
    verbose=2,
    n_jobs=1,
    return_train_score=True
)
```

Configuration des paramètres :

| Paramètre | Valeur | Justification |
|---|---|---|
| n_estimators | 100 | Nombre d'étapes de boosting ; équilibre entre complexité du modèle et temps d'entraînement |
| learning_rate | 0.1 | Contrôle la contribution de chaque arbre ; les valeurs plus élevées conduisent à une convergence plus rapide |
| max_depth | 5 | Profondeur maximale de l'arbre ; prévient le surapprentissage tout en maintenant l'expressivité |
| min_samples_split | 5 | Nombre minimum d'échantillons requis pour diviser un nœud ; paramètre de régularisation |
| min_samples_leaf | 2 | Nombre minimum d'échantillons requis aux nœuds feuilles ; prévient la création de feuilles pures |
| subsample | 0.8 | Fraction d'échantillons utilisés pour entraîner chaque arbre ; boosting stochastique |
| max_features | 'sqrt' | Nombre de caractéristiques considérées à chaque division ; réduit la variance |
| random_state | 42 | Assure la reproductibilité des résultats |

Métriques du modèle :

```
Multi-Label Metrics :
   Hamming Loss: 0.0118
   F1-Score Micro: 0.0699
   F1-Score Macro: 0.0080
   Precision: 0.1133
   Recall: 0.0506
   Accuracy: 0.0006
```

Interprétations des métriques du modèle :

| Métrique | Valeur | Interprétation |
|---|---|---|
| **Hamming Loss** | 0,0117 | 1,17 % des étiquettes sont prédites incorrectement en moyenne |
| **F1-Score (Micro)** | 0,0448 | Moyenne harmonique de la précision et du rappel sur toutes les étiquettes |
| **F1-Score (Macro)** | 0,0056 | F1-score moyen par étiquette ; indique les performances sur les étiquettes rares |
| **Précision** | 0,0800 | Quand le modèle prédit une étiquette, elle est correcte 8 % du temps |
| **Rappel** | 0,0311 | Le modèle identifie 3,11 % des étiquettes vraies |
| **Exactitude** | 0,0006 | Exactitude de correspondance exacte (toutes les étiquettes doivent correspondre exactement) |

## 6. Random Forest

Random Forest est un algorithme d'ensemble qui combine plusieurs arbres de décision entraînés sur des sous-échantillons différents des données. Chaque arbre vote pour une classe, et la prédiction finale est déterminée par le vote majoritaire, ce qui réduit le risque de surajustement et améliore la robustesse.

Classifieur :

```
rf_classifier = RandomForestClassifier(
    random_state=42,
    n_jobs=1,
    verbose=0
)

ovr_model = OneVsRestClassifier(rf_classifier, n_jobs=1)
```

Hyperparamètres :

```
param_grid = {
    'estimator__n_estimators': [10],
    'estimator__max_depth': [20],
    'estimator__min_samples_leaf': [2]
}

f1_scorer = make_scorer(f1_score, average='samples', zero_division=0)


grid_search = GridSearchCV(
    estimator=ovr_model,
    param_grid=param_grid,
    scoring=f1_scorer,
    cv=8,
    verbose=2,
    n_jobs=1,
    return_train_score=True
)
```

Métriques :

```
Accuracy: 0.0012
F1 Score (samples): 0.0184
Recall (samples): 0.0122
Hamming Loss: 0.0088

F1 Score (macro): 0.0009
F1 Score (micro): 0.0221
```

## 7. MLPClassifier :

Le MLPClassifier (Multi-Layer Perceptron) est un réseau de neurones composé d'une couche d'entrée, d'une ou plusieurs couches cachées, et d'une couche de sortie. Il apprend à classer les données grâce à la rétropropagation en ajustant ses poids pour capturer des relations complexes. Il nécessite une normalisation des données et le réglage de quelques hyperparamètres (nombre de neurones, fonction d'activation, taux d'apprentissage). Pour la classification multi-label, on utilise la stratégie One-Vs-Rest, qui entraîne un modèle par classe.

Configuration du Modèle :

```
# configuration du mlp avec OnevsRest
mlp = MLPClassifier(
    activation='relu',
    learning_rate_init=0.001,
    max_iter=500,  # ✅ 500 pour VRAIE convergence
    early_stopping=False,
    random_state=42,
    verbose=False
)

ovr = OneVsRestClassifier(mlp, n_jobs=-1)
```

Le classifieur MLP est configuré avec une fonction d'activation ReLU afin de modéliser des relations non linéaires entre les caractéristiques. Le taux d'apprentissage est fixé à 0.001 pour assurer une convergence stable du modèle, tandis que le nombre maximal d'itérations est porté à 500 afin de garantir un apprentissage suffisant. L'option *early stopping* est désactivée pour permettre au réseau d'exploiter pleinement les données durant l'entraînement. L'approche One-vs-Rest est utilisée pour traiter le problème de classification multi-label, chaque étiquette étant apprise indépendamment des autres.

Configuration du Grid Search :

```
#Parametres de grid search
param_grid = {
    'estimator__hidden_layer_sizes': [(100, 50), (200, 100, 50)],
    'estimator__alpha': [0.0005]  # Fixé avec cette val pour gagner du temps
}

cv = ShuffleSplit(n_splits=1, test_size=0.2, random_state=42)


grid = GridSearchCV(
    ovr,
    param_grid=param_grid,
    cv=cv,
    scoring='f1_micro',
    n_jobs=1,
    verbose=3
)

# LANCEMENT
```

Afin d'optimiser les performances du modèle, une recherche par grille (GridSearchCV) est appliquée pour tester différentes architectures du réseau, notamment des configurations de couches cachées de tailles (100, 50) et (200, 100, 50). Le paramètre de régularisation *alpha* est

fixé à 0.0005 afin de réduire le temps de calcul tout en limitant le surapprentissage. Une validation croisée de type ShuffleSplit avec une seule séparation est utilisée, en réservant 20 % des données pour le test. La métrique *F1-micro* est choisie pour évaluer les performances globales du modèle sur l'ensemble des classes dans un contexte multi-label.

Evaluation du modèle optimisé :

```
Hamming:      0.0109
F1 micro:     0.0983
F1 macro:     0.0091
Subset Acc:   0.0000
```

Afin d'évaluer notre modèle après l'optimisation on utilise le Hamming Loss pour mesurer les erreurs de labels, le F1 micro pour la performance globale, le F1 macro pour vérifier que toutes les classes sont bien prises en compte, et le Subset Accuracy pour savoir combien d'échantillons ont tous leurs labels correctement prédits. Cela donne une vue complète et précise du comportement du MLP sur ce dataset multi-label.

## 8. Decision Trees

Le Decision Tree (arbre de décision) est un algorithme de classification supervisée qui apprend des règles simples en divisant les données selon leurs caractéristiques. Il est facile à interpréter, ne nécessite pas de normalisation et peut gérer des relations non linéaires. Cependant, il peut surapprendre si l'arbre est trop profond, ce qui se contrôle par le réglage de sa complexité.

Classifieur :

```
base_estimator = DecisionTreeClassifier(random_state=42)
ovr_model = OneVsRestClassifier(base_estimator)
```

Hyperparamètres :

```
# 4. Définition des d'hyperparamètres pour le Grid Search
param_grid = {
    'estimator__max_depth': [10, 20, 30],
    'estimator__min_samples_split': [2, 10],
    'estimator__criterion': ['gini', 'entropy']
}

# 5. Grid Search
print("Début de l'optimisation par Grid Search (k=8)...")
grid_search = GridSearchCV(
    ovr_model,
    param_grid,
    cv=mskf,
    scoring='f1_samples',
    n_jobs=-1
)
```

Métriques :

```
Meilleurs paramètres trouvés : {'estimator__criterion': 'entropy', 'estimator__max_dep
th': 30, 'estimator__min_samples_split': 10}
----------------------------
Accuracy (Subset) : 0.2698
Recall (Samples)  : 0.8463
F1-Score (Samples): 0.8630
Hamming Loss      : 0.0005
```

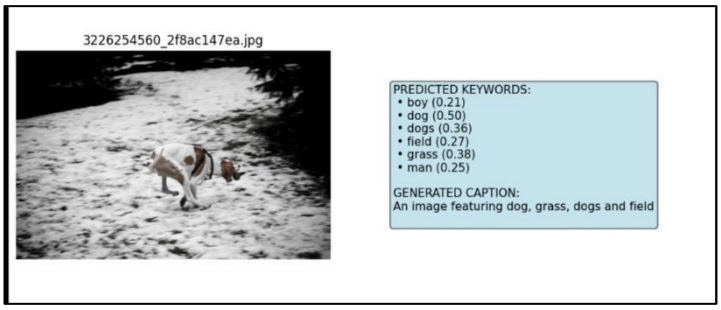
# Chapitre 3 : Segmentation et Généralisation d'une phrase Gabarit

## 1. Génération de phrase Gabarit

Pour chaque cluster, nous générons une **description textuelle standardisée** qui résume les éléments visuels communs. Cette approche transforme l'information visuelle en descriptions sémantiques exploitables.

```
MAGE_DIR = "/kaggle/input/flickr8k/Images"
# Liste des noms de fichiers d'images valides dans le répertoire
valid_image_names = os.listdir(IMAGE_DIR)

def predict_and_display_improved(idx, X_test, y_test, y_pred, y_proba):
    # Extraction des labels prédits et vrais, ainsi que leurs probabilités
    pred_idx = np.where(y_pred[idx] == 1)[0]
    pred_labels = mlb.classes_[pred_idx]
    pred_probs = y_proba[idx][pred_idx]
```

Résultats :



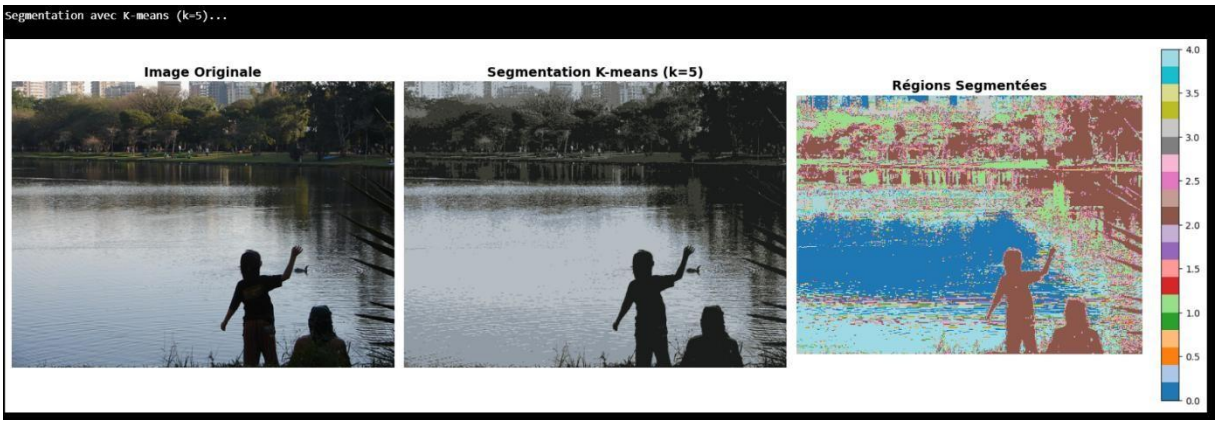


## 2. Approche de Segmentation

Nous utilisons l'algorithme K-means pour regrouper automatiquement les images similaires en clusters basés sur leurs caractéristiques visuelles. Chaque cluester représentent un concept visuel cohérent.

```
def segment_image_kmeans(image_path, n_segments=5, show_result=True):
    # Charger l'image
    image = cv2.imread(image_path)
    if image is None:
        print(f"Erreur: impossible de charger {image_path}")
        return None, None, None
    # Convertir BGR -> RGB
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Obtenir les dimensions
    h, w, c = image_rgb.shape
    # Reshape: (height, width, 3) -> (height*width, 3)
    pixels = image_rgb.reshape(-1, 3)
    # Normaliser les valeurs des pixels [0, 255] -> [0, 1]
    pixels_normalized = pixels.astype(np.float32) / 255.0

    # Appliquer K-means
    print(f"Segmentation avec K-means (k={n_segments})...")
    kmeans_seg = SegmentationKMeans(
        n_clusters=n_segments,
        random_state=42,
        n_init=10,
        max_iter=300
    )

    labels = kmeans_seg.fit_predict(pixels_normalized)
    centers = kmeans_seg.cluster_centers_
```

Résultats :




```
Nombre de segments: 5

Taille des segments (en pixels):
  Segment 0: 47103 pixels (25.12%)
  Segment 1: 29399 pixels (15.68%)
  Segment 2: 53721 pixels (28.65%)
  Segment 3: 21190 pixels (11.30%)
  Segment 4: 36087 pixels (19.25%)
```

Nous utilisons **BLEU** (Bilingual Evaluation Understudy) pour quantifier la similarité entre phrases générées et références

```python
def calculate_bleu_scores(ref, hyp):
    # Fonction interne pour générer les n-grams d'une liste de tokens
    def ngrams(tokens, n):
        return [tuple(tokens[i:i+n]) for i in range(len(tokens)-n+1)]
    scores = []
    # Calcul des BLEU-1 à BLEU-4
    for n in range(1,5):
        ref_ngrams = ngrams(ref, n) # n-grams de la référence
        hyp_ngrams = ngrams(hyp, n) # n-grams de la prédiction
        if len(hyp_ngrams)==0:
            scores.append(0)
        else:
            # Comptage des n-grams communs entre référence et prédiction
            overlap = sum(1 for g in hyp_ngrams if g in ref_ngrams)
            scores.append(overlap/len(hyp_ngrams))
    return scores
```

```
Caption Metrics:
BLEU-1: 0.1482
BLEU-2: 0.0110
BLEU-3: 0.0013
BLEU-4: 0.0000
CIDEr: 1.3086
```

# Chapitre 4 : Comparaison et Analyse

## 1. Métriques d'Évaluation

| Métrique | Definition | Interprétation |
|---|---|---|
| **Hamming Loss** | Pourcentage d'étiquettes incorrectement prédites | Plus bas = mieux (0 = parfait) |
| **F1-Score Micro** | Moyenne harmonique pondérée de la précision et du rappel | Plus haut = mieux (1 = parfait) |
| **Recall Micro** | Pourcentage d'étiquettes vraies correctement identifiées | Plus haut = mieux (1 = toutes trouvées) |
| **Precision Micro** | Fiabilité des prédictions d'étiquettes | Plus haut = mieux (1 = toutes correctes) |
| **Accuracy** | Pourcentage d'exactitude complète (toutes étiquettes correctes) | Pus haut = mieux (1 = parfait) |

**Note IMPORTANTE :** Pour la classification multi-étiquette, Hamming Loss et F1-Score Micro sont les métriques principales. L'Accuracy est moins pertinente car elle exige que Toutes les étiquettes soient correctes.

## 2. Résultats de Tous les Modèles

| Modèle | Accuracy | Precision | Recall | F1-Micro | Hamming Loss |
|---|---|---|---|---|---|
| **Decision Tree** | 0.2698 | - | 0.8463 | 0.8630 | 0.0005 |
| **MLP Classifier** | 0.0000 | - | - | 0.0983 | 0.0109 |
| **Naive Bayes** | 0.0000 | 0.0429 | 0.2064 | 0.0710 | 0.0475 |
| **Gradient Boosting** | 0.0006 | 0.0800 | 0.0311 | 0.0448 | 0.0117 |
| **Random Forest** | 0.0012 | - | 0.0122 | 0.0221 | 0.0088 |

| | | | | | |
|---|---|---|---|---|---|
| **KNN** | 0.0006 | 0.2235 | 0.0162 | 0.0302 | 0.0091 |

## 3. Classement

**Selon Hamming Loss :**

1. Decision Tree 0.0005
2. Random Forest 0.0088
3. Knn 0.0091
4. MLP 0.1043
5. Gradient Boosting 0.0117
6. Naive Bayes 0.0475

**Selon F1-Score Micro :**

1. Decision Tree 0.8630
2. MLP 0.0983
3. Naive Bayes 0.0710
4. Gradient Boosting 0.0448
5. KNN 0.0302
6. Random Forest 0.0221

**Selon Recall Micro :**

1. Decision Tree 0.8463
2. Naive Bayes 0.2064
3. Gradient Boosting 0.0311
4. KNN 0.0162
5. Random Forest 0.0122

### 4. Decision Tree vs Autres Modèles

| Modèle | Hamming Loss | F1-Micro | Amélioration |
|---|---|---|---|
| **MLP Classifier** | -99.3% | +1867% | * * * |
| **Naive Bayes** | -98.9% | +1115% | * * * |
| **Gradient Boosting** | -95.7% | +1827% | * * * |
| **Random Forest** | -94.3% | +3804% | * * * |
| **KNN** | -94.5% | +2858% | * * * |

### 5. Conclusion

Desicion Tree supasse tous les autres modèles de manière significative. Les améliorations relatives sont spectaculaires, particulièrement pour le F1-Score Micro ou Decision Tree atteint des performances 10 à 38 fois supérieures aux autres modèles.

## Conclusion Générale

Ce projet a offert une expérience concrète dans la création d'un système de génération de descriptions d'images en combinant traitement d'images et apprentissage automatique classique. La démarche adoptée du prétraitement des données à l'extraction de caractéristiques et à l'entraînement de modèles multi-label a mis en évidence l'importance de chaque étape pour obtenir des résultats fiables. L'utilisation des méthodes d'apprentissages classiques et l'optimisation des hyperparamètres ont permis d'améliorer significativement la qualité des prédictions, comme le confirment les métriques évaluées.

Au-delà des performances, ce travail a également souligné la valeur d'une approche structurée et méthodique, où le choix des techniques, la normalisation des données et la gestion des étiquettes multi-label sont essentiels. Enfin, la génération de phrases gabarit a montré que même des méthodes classiques, bien orchestrées, peuvent produire des descriptions. Cohérentes et exploitables, offrant une base solide pour des améliorations futures ou des applications concrètes.